Naval Research Laboratory

Washington, DC 20375-5000

NRL Memorandum Report 6216

A Trace Specification of the MMS Security Model

CHARLES B. CROSS

Formal Methods Section
Information Technology Division

July 8, 1988

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public releases; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NRL Memorandum Report 6216	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Research Laboratory 6b. OFFICE SYMBOL (If applicable) Code 5590	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Washington, DC 20375-5000	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) A Trace Specification of the MMS Security Model	
12. PERSONAL AUTHOR(S) Cross, Charles B.	
13a. TYPE OF REPORT 13b. TIME COVERED FROM 1/87 TO 9/87 14. DATE OF REPORT (Year, Month, Day) 1988 July 8 15. PAGE COUNT 32	
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Logic, Verification, Program semantics, Trace specification, Specification

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The MMS security model is an abstract, mathematically precise specification of the information security requirements of a multilevel secure electronic message system. This report presents a trace specification of the basic behavior and security properties of thirty one procedures that might form part of such a system. These procedures are specified in such a way as to guarantee that they satisfy the requirements of the MMS security model, given an appropriate trace interpretation of the predicates found in the MMS model. The specification is written using the Hoffman trace specification heuristics, but normal forms are defined recursively rather than explicitly.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Charles B. Cross	22b. TELEPHONE (Include Area Code) (202) 767-3490 22c. OFFICE SYMBOL Code 5590

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

CONTENTS

1 INTRODUCTION 1

2 THE MMS MODEL TRACE SPECIFICATION 3

REFERENCES 21

APPENDIX — The MMS Model Specification 23

A TRACE SPECIFICATION OF THE MMS SECURITY MODEL*

1 Introduction

The MMS security model presented in [Land84] is an abstract, mathematically precise specification of the information security requirements of a multilevel secure electronic message system. This report presents a trace specification of the basic behavior and security properties of thirty one procedures that might form part of such a message system. These procedures are specified in such a way as to guarantee that they satisfy the requirements of the security model presented in [Land84], given a certain trace interpretation of the predicates found in the MMS model.

The remainder of this section contains background material and may be skimmed or skipped by the reader who is already familiar with the trace specification language and the Hoffman methodology.

1.1 The Trace Specification Language

The trace language provides for the specification of software modules in terms of the effects (such as return values) that the user sees when (s)he executes a sequence of procedure and function calls. These sequences are called *traces*. The idea of basing specifications on traces was first suggested in [Bart77] and later formalized in [McLean85]. The heuristic methodology presented in [Hoff84] and [Hoff86] makes it easier to understand and write trace specifications, and I will adopt that methodology here.

A trace specification consists of a *syntax* section and a *semantics* section. The syntax section states the name and parameter types of each of the module's procedures and the name, parameter types, and return value type of each of the module's function calls. The semantics section contains axioms formalized in a two-sorted language of first-order logic with identity, with one set of variables $\{R, R_1, R_2, \ldots S, S_1, S_2, \ldots T, T_1, T_2, \ldots\}$ to be understood as ranging over traces. In addition to the usual logical connectives there is an interpreted binary function symbol (.), which serves as a notation for concatenating trace terms. If X is a trace variable, the empty trace e, a procedure call, or a function call, then X is a well-formed trace term; if X and Y are trace terms, then $(X.Y)$ is a well-formed trace term. Nothing else is a trace term. A function (procedure) call is a function (procedure) name followed by the requisite number of parameters of appropriate types. In place of a formal axiom of associativity for concatenation I adopt the convention of dropping the parentheses around the subterms of a trace term.

Manuscript approved February 4, 1988.

*Thanks to John McLean, Carl Landwehr, Andy Moore, and Mark Cornwell for useful comments along the way.

The axioms that appear in the semantics section of a trace specification state or entail information about which traces are *legal* and about the values returned by legal traces that end with function calls. The legality predicate and the value function are usually formalized using the unary predicate symbol L and the unary function symbol V, respectively. One additional and very handy piece of notation is *trace equivalence* $\equiv$, defined as follows ([McLean86], p. 4):

$$S \equiv T =_{df} \forall R\,[(L(S.R) \leftrightarrow L(T.R)) \wedge (R \neq e \leftrightarrow (\exists x\, V(S.R) = x \leftrightarrow V(S.R) = V(T.R)))]$$

In other words, two traces are equivalent just in case they agree on (i) present and future legality and (ii) all future return values. Intuitively, two traces are equivalent provided that they place the module in the same "state," as far as the user can tell. The intuitive idea of module state has great heuristic value and plays an important role in the Hoffman methodology, which I will presently describe.

1.2 The Hoffman Methodology

In [Hoff84] (and in [Hoff86] with Richard Snodgrass) Daniel Hoffman describes a set of heuristics for writing trace specifications, the most basic of which being these:[1]

- Base the specification on a definition of normal form traces.
- Structure the semantics according to normal form prefixes.

The notion of a *normal form* trace is based on the following concept of *module state*: Two traces place the module into the same state provided that a user could not distinguish between them by appending a series of procedure and function calls and comparing return values (and error messages). In other words, two traces represent the same state iff they are trace-equivalent. *Normal form* traces are the canonical representatives of the states of a module. They allow one to structure a trace specification in such a way as to make explicit exactly how procedures move the module from one state to another. Formally, a normal form for a trace specification is simply any set of traces containing at least one trace from each equivalence class of the trace-equivalence relation $\equiv$. Every legal trace is therefore equivalent to at least one normal form trace, for any given normal form.

One would typically define a normal form by means of an *explicit* definition. For example, in his simple stack specification Hoffman defines a trace to be in normal form iff it consists only of **push** calls.[2] It is also possible to define a normal form recursively. One does this in two steps: first, by stipulating that the empty trace is in normal form; then, for each normal form trace T and procedure call C, one states the conditions under which $T.C$ is in normal form. Since one is normally interested only in legal traces, one can instead do the recursion by stating, for each normal form trace T and procedure call C such that $T.C$ is a legal trace, the conditions under which $T.C$ is a normal form trace.

Given a definition of normal form and trace-legality, the next step, according to the Hoffman methodology, is to specify an equivalent normal form trace for each legal trace $T.C$ such that T

[1] [Hoff86], p. 6.

[2] See [Hoff86], p. 8.

is in normal form but $T.C$ is not. Doing this insures that every legal trace is provably equivalent to some normal form trace; consider an arbitrary legal trace $C_1.C_2 \ldots C_n$. If every legal trace of the form $T.C$, where T is in normal form, is provably equivalent to some normal form trace, then a normal form trace provably equivalent to $C_1.C_2 \ldots C_n$ can be constructed as follows: let $S_0 = e$, and if $S_i.C_{i+1}$ is a normal form trace and $i < n$, then let $S_{i+1} = S_i.C_{i+1}$, otherwise let $S_{i+1} = T$, where T is a normal form trace equivalent to $S_i.C_{i+1}$. Each S_i is in normal form, so there will always exist such a T. Clearly, S_n is trace-equivalent to $C_1.C_2 \ldots C_n$.

Once a normal form equivalent has been specified for each trace having a normal form prefix, the final step is to specify the return values of all function calls. This is done by specifying the value of every legal trace $T.C$, where T is any normal form trace and C is any function call.

2 The MMS model trace specification

This section begins with a discussion of the way in which normal forms and legality are defined in the MMS model trace specification. Following this will be a discussion of the procedures that have been specified, grouped by function, and a discussion of the predicates that define the security properties of the MMS model.

2.1 Normal forms and legality

Normal forms and legality in the MMS model specification are defined by mutually recursive axioms of the following form for each procedure call in the module:

$$nf(T) \rightarrow (\mathrm{L}(T.\langle \textbf{call} \rangle) \leftrightarrow \langle \textbf{formula1} \rangle)$$

$$(\mathrm{nf}(T) \wedge \mathrm{L}(T.\langle \textbf{call} \rangle)) \rightarrow (\mathrm{nf}(T.\langle \textbf{call} \rangle) \leftrightarrow \langle \textbf{formula2} \rangle)$$

where the formulas that replace $\langle$**formula1**$\rangle$) and $\langle$**formula2**$\rangle$) are assumed not to contain any occurrence of the predicates nf and L. In addition, we assume that the empty trace e is a normal form trace, that the longest proper prefix of a legal trace is legal, that every normal form trace is legal, and that the longest proper prefix of a normal form trace is a normal form trace; i.e. we assume:

$$\mathrm{nf}(e)$$

$$\forall C\,(\mathrm{L}(T.C) \rightarrow \mathrm{L}(T))$$

$$(\mathrm{nf}(T) \rightarrow \mathrm{L}(T))$$

$$\forall C\,(\mathrm{nf}(T.C) \rightarrow \mathrm{nf}(T))$$

These two schemata and the fouradditional axioms displayed above are sufficient to define the legality and normal form status of every trace. To see that this is so, consider the following construction of the respective truth values of nf(T) and L(T) for an arbitrary trace T: If $T = e$, then nf(T) and L(T) are true, and we are done. Suppose instead that $T = R.C$ and that the truth values of nf(R) and L(R) are known. If L(R) is false, then nf(T) and L(T) are both false, and we are done. Suppose, then, that L(R) is true, and define the trace S as follows:

$$S = \begin{cases} R, & \text{if } \mathrm{nf}(R); \\ S', & \text{otherwise, for some } S' \text{ where } S' \equiv R \text{ and } \mathrm{nf}(S'). \end{cases}$$

By the axiom schemata, there exist formulas ν and λ such that the following are axioms:

$$\mathrm{nf}(S) \rightarrow (\mathrm{L}(S.C) \leftrightarrow \psi)$$

$$(\mathrm{nf}(S) \wedge \mathrm{L}(S.C)) \rightarrow (\mathrm{nf}(S.C) \leftrightarrow \chi)$$

Given these formulas the truth values of nf(T) and L(T) are determined as follows: L(T) has the same truth value as ψ, since $S \equiv R$; if L($S.C$) is false, then by the equivalence of S and R, L(T) is also false and so nf(T) is false. Suppose L($S.C$) is true; if $S \neq R$, then nf(R) is false and so nf(T) is false. If $S = R$, then $T = S.C$, and so nf(T) has the same truth value as χ.

This construction shows that legality and the normal form predicate are completely "defined" in the following sense: given axioms of the sort described above as hypotheses, the truth value of any formula of the form nf(T) or of the form L(T) is provably a function of the truth values of formulas that do not contain the normal form and legality predicates. The construction is not an effective procedure for *deciding* normal form-hood and legality, however, because ψ and χ may not be decidable formulas and because the formula S' is merely chosen, not constructed.

2.2 Initializing procedures

First we consider the procedures **create_user**, **delete_user**, **login**, and **logout**; **create_user** is specified as follows:[3]

Create_user

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathbf{create\_user}(u, v, l)) \leftrightarrow [\mathrm{RO}(u, T, sso) \wedge \mathrm{user\_exists}(u, T) \wedge \mathrm{logged\_in}(u, T) \wedge$
$\neg \mathrm{user\_exists}(v, T) \wedge \mathrm{ref\_secure}(T, \mathbf{create\_user}(u, v, l))]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{create\_user}(u, v, l))] \rightarrow [\mathrm{nf}(T.\mathbf{create\_user}(u, v, l) \leftrightarrow \neg \exists u' \exists R\, [T = R.\mathbf{delete\_user}(u', v) \wedge \mathrm{CU}(v, R) = l]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{create\_user}(u, v, l)) \wedge T = R.\mathbf{delete\_user}(u, v) \wedge \mathrm{CU}(v, R) = l] \rightarrow T.\mathbf{create\_user}(u, v, l) \equiv R$

These formulas state the following: first, the result of appending **create_user** to a normal form trace is legal if and only if: the creating user exists, is logged in, and has the role of *System Security Officer*, the created user does not already exist, and the trace and procedure call also stand in the ref_secure relation, about which we shall have more to say later. If these conditions are satisfied, then this trace, which results from appending **create_user** to a normal form trace T, is itself in normal form just in case T does not end with a deletion of the soon-to-be-created user while he has the same security clearance that he is about to be created at. The third axiom states that if the user *is* redundantly deleted and then recreated at the same security clearance, it is then as if neither operation had been performed.

Obviously, one user must exist at the beginning in order to create the others. This user is called *Root* in the specification; *Root* exists by definition, and is defined to have the role of system security officer at all times.

These are the definitions of all but one of the predicates used in the specification of **create_user**. (The definition of the one other predicate, ref_secure, can be found in the Appendix. Its definition presupposes the material introduced in section 2.4.)

$\mathrm{RO}(v, T, r) \rightarrow [[v = Root \wedge r = sso] \vee \exists S\, [\mathrm{some\_none}_{(2,2)}(S, \mathbf{add\_RO}, \mathbf{rm\_RO}, 2, v, 3, r; 2, v, 3, r, T) \wedge$
$\neg \mathrm{in\_after}_1(S, \mathbf{delete\_user}, 2, v, T)]]$

[3] For an explanation of the trace predicates used here and throughout the rest of this report, see [Land84].

$\text{user\_exists}(u, T) \leftarrow [u = Root \vee \exists S\ \text{some\_none}_{(1,1)}(S, \textbf{create\_user}, \textbf{delete\_user}, 2, u; 2, u, T)]$

The in_after and some_none predicates are convenient abbreviations for very long and complicated formulas[4] and will be used many times. some_none is really a family of predicates that allow one to say that a trace contains an occurrence of a procedure call of one sort that is not followed by an occurrence of a procedure call of a certain other sort. More specifically, $\text{some\_none}_{(m,n)}(S, c_1, c_2, j_1, x_1, \ldots, j_m, x_m; k_1, y_1, \ldots, k_n, y_n, T)$ states that S is a prefix of T and is followed in T by a call to procedure c_1. This call has argument x_i in position j_i, for i from 1 to m_i, and there does not exist in T after S and the call to c_1 that follows it, any occurrence of a call to c_2 having argument y_i in position k_i, for i from 1 to n. The predicate in_after has a similar flavor: $\text{in\_after}_n(S, c, k_1, x_1, \ldots, k_n, x_n, T)$ states that a call to procedure c with x_i as the k_ith argument, for i from 1 to n, occurs immediately after S in T, where, again S is a prefix of T. These predicates, and similar predicates discussed later on, are useful because they allow one to specify the current value of some parameter, for example, someone's security clearance, as the value it was given *the last time it was modified.*

The procedure **delete_user** has this specification:

Delete_user

$\text{nf}(T) \rightarrow [\text{L}(T.\textbf{delete\_user}(u, v)) \leftrightarrow$
$[\text{logged\_in}(u, T) \wedge \text{RO}(u, T, sso) \wedge \text{user\_exists}(v, T) \wedge \text{ref\_secure}(T, \textbf{delete\_user}(u, v))]]$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{delete\_user}(u, v)) \rightarrow [\text{nf}(T.\textbf{delete\_user}(u, v) \leftrightarrow [\neg\exists u'\ \exists S\ T = S.\textbf{create\_user}(u', v)]$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{delete\_user}(u, v)) \wedge T = S.\textbf{create\_user}(u', v)] \rightarrow T.\textbf{delete\_user}(u, v) \equiv S]$

Note that with **delete_user**, as with **create_user**, normal form status depends on the absence of procedure calls that immediately cancel one another out. A trace ending with a call to **delete_user** is in normal form only if its longest proper prefix does not end with a call to **create_user** that creates the userid that is about to be deleted.

Next we consider the **login** and **logout** procedures. They are specified as follows:

login

$\text{nf}(T) \rightarrow [\text{L}(T.\textbf{login}(u, d)) \leftrightarrow [\neg\text{logged\_in}(u, d, T) \wedge \text{user\_exists}(u, T) \wedge \text{ref\_secure}(T, \textbf{login}(u, d))]]$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{login}(u, d))] \rightarrow [\text{nf}(T.\textbf{login}(u, d)) \leftrightarrow \neg\exists x\ \exists R\ T = R.\textbf{logout}(u, d)]$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{login}(u, d)) \wedge T = R.\textbf{logout}(u, d))] \rightarrow T.\textbf{login}(u, d) \equiv T$

logout

$\text{nf}(T) \rightarrow [\text{L}(T.\textbf{logout}(u, d)) \leftrightarrow [\text{logged\_in}(u, d, T) \wedge \neg\exists r\ \exists k\ \text{H}(d, T, r, k) \wedge \text{ref\_secure}(T, \textbf{logout}(u, d))]]$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{logout}(u, d))] \rightarrow [\text{nf}(T.\textbf{logout}(u, d)) \leftrightarrow \neg\exists S\ T = S.\textbf{login}(u, d)]$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{logout}(u, d)) \wedge T = S.\textbf{login}(u, d)] \rightarrow T.\textbf{logout}(u, d) \equiv S]$

That is, logging in at a given device d is legal only if you are not already logged in on d, and logging in preserves normal form-hood only if you did not just logout of the same device.

[4] See the definitions in the appendix.

Similarly, logging out is legal only if you are logged in, and logging out preserves normal form-hood only if you did not just login to the same device. The property of being logged in is specified as follows:

$$\text{logged\_in}(u, d, T) \leftrightarrow \exists R\,[\text{user\_exists}(u, T) \wedge \text{some\_none}_{(2,2)}(R, \textbf{login}, \textbf{logout}, 1, u, 2, d; 1, u, , 2, d, T)]$$

That is, a user is logged in at a device only if there is a point in the past at which an appropriate call to **login** was not followed by a corresponding call to **logout**.

2.3 Classifications and clearances

The next group of procedures that we examine are those dealing with the various security parameters associated with users and entities. In addition to procedures that set values and modify lists of values, there are functions, available only to a system security officer, that return the values of security variables.

2.3.1 Setting values

Certain security parameters are single values. These are: the security clearance of a user, the maximum classification of an output device, the actual classification of an entity, and the CCR value of a container. These parameters are set using the procedures **set_cu**, **set_CD**, **set_CE**, and **set_CCR**, respectively. The specifications for these procedures are all based on the idea that a parameter setting procedure preserves normal form-hood if and only if it neither sets a parameter to the value it already has nor cancels out the effect of a parameter change that has just taken place. So we have, for example, the specification of **set_cu**:

set_CU

$$\text{nf}(T) \rightarrow [\text{L}(T.\textbf{set\_cu}(u, v, l)) \rightarrow [\text{RO}(u, T, sso) \wedge \text{user\_exists}(u, T) \wedge \text{logged\_in}(u, T) \wedge \text{user\_exists}(v, T) \wedge \neg\text{logged\_in}(v, T) \wedge \text{ref\_secure}(T.\textbf{set\_cu}(u, v, l))]]$$

$$[\text{nf}(T) \wedge \text{L}(T.\textbf{set\_cu}(u, v, l))] \rightarrow [\text{nf}(T.\textbf{set\_cu}(u, v, l)) \rightarrow [\text{CU}(v, T) \neq l \wedge \neg\exists S\, \exists u'\, \exists v'\, \exists l'\, T = S.\textbf{set\_cu}(u', v', l')]$$

$$[\text{nf}(T) \wedge \text{L}(T.\textbf{set\_cu}(u, v, l)) \wedge \text{CU}(v, T) = l] \rightarrow T.\textbf{set\_cu}(u, v, l) \equiv T$$

$$[\text{nf}(T) \wedge \text{L}(T.\textbf{set\_cu}(u, v, l)) \wedge T = S.\textbf{set\_cu}(u', v', l')] \rightarrow T.\textbf{set\_cu}(u, v, l) \equiv S.\textbf{set\_cu}(u, v, l)]$$

Note the last two formulas in the specification. They specify *different* normal form equivalents for the trace $T.\textbf{set\_cu}(u, v, l)$ depending on the particular way in which it fails to be in normal form.

Procedures that manipulate the security properties of entities are specified in much the same way. Since trace specifications give the user's view of procedure behavior, we will specify the security parameters of entities as properties of the *references* that refer to them. We begin with **set_CD**, a procedure for setting the maximum classification of information allowed to appear on a given output device.

set_CD

$$\text{nf}(T) \rightarrow [\text{L}(T.\textbf{set\_CD}(u, d, l)) \rightarrow$$
$$[\text{logged\_in}(u, T) \wedge \text{RO}(u, T, sso) \wedge \text{CE}(d, T) \leq l \wedge \text{device}(d) \wedge \text{ref\_secure}(T.\textbf{set\_CD}(u, d, l))]]$$

$[nf(T) \wedge L(T.\mathbf{set\_CD}(u, d, l))] \rightarrow [nf(T.\mathbf{set\_CD}(u, d, l)) \leftrightarrow [\neg\exists u'\, \exists l'\, \exists S\, T = S.\mathbf{set\_CD}(u', d, l') \wedge$
$\forall C\, \forall R\, [[prefix(R.C, T) \wedge callname(C) = \mathbf{set\_CD} \wedge coref(arg(C, 2), R, d, T)] \rightarrow [arg(C, 2) \neq l \vee \exists C'\, \exists S\, [callname(C') =$
$\mathbf{set\_CD} \wedge coref(arg(C', 2), S, d, T) \wedge prefix(R.C, S.C') \wedge arg(C', 2) \neq l]]]]$

$[nf(T) \wedge L(T.\mathbf{set\_CD}(u, d, l)) \wedge T = S.\mathbf{set\_CD}(u', d, l')] \rightarrow T.\mathbf{set\_CD}(u, d, l) \equiv S.\mathbf{set\_CD}(u, d, l)$

$[nf(T) \wedge L(T.\mathbf{set\_CD}(u, d, l)) \wedge \exists C\, \exists R\, [prefix(R.C, T) \wedge callname(C) = \mathbf{set\_CD} \wedge coref(arg(C, 2), R, d, T) \wedge arg(C, 2) = l \wedge$
$\neg\exists C'\, \exists S\, [callname(C') = \mathbf{set\_CD} \wedge coref(arg(C', 2), S, d, T) \wedge prefix(R.C, S.C') \wedge arg(C', 2) \neq l]] \rightarrow T.\mathbf{set\_CD}(u, d, l) \equiv$
T

The very complicated looking second formula in the specification for this procedure can be explained as follows: a legal call to **set_CD** preserves normal form-hood just in case (i) it does not immediately cancel the effect of another **set_CD** call, and (ii) any previous **set_CD** call that set the maximum classification of the device to the same level that this call sets it to was cancelled by a subsequent **set_CD** call setting it to a different classification. In case (i) fails, the trace is equivalent (by the third formula) to the result of deleting the **set_CD** call whose effect was cancelled. If (ii) fails, the the trace is equivalent (by the fourth formula) to the result of deleting the final **set_CD** call.

The four-place relation coref appears for the first time in the specification of this procedure. Its intended meaning is this: $coref(r_1, S, r_2, T)$ states that r_1 refers at S to the same entity that r_2 refers to at T. This predicate will be discussed further in the section on references.

The specification of **set_CE** is very similar to that of **set_CD**:

set_CE

$nf(T) \rightarrow [L(T.\mathbf{set\_CE}(u, d, l)) \leftrightarrow$
$[logged\_in(u, T) \wedge AS(u, \mathbf{set\_CE}, d, 2, T) \wedge device(d) \wedge l \leq CD(d, T) \wedge ref\_secure(T, \mathbf{set\_CE}(u, d, l))]]$

$[nf(T) \wedge L(T.\mathbf{set\_CE}(u, d, l))] \rightarrow [nf(T.\mathbf{set\_CE}(u, d, l)) \leftrightarrow [\neg\exists u'\, \exists l'\, \exists S\, T = S.\mathbf{set\_CE}(u', d, l') \wedge$
$\forall C\, \forall R\, [[prefix(R.C, T) \wedge callname(C) = \mathbf{set\_CE} \wedge coref(arg(C, 2), R, d, T)] \rightarrow [arg(C, 2) \neq l \vee \exists C'\, \exists S\, [callname(C') =$
$\mathbf{set\_CE} \wedge coref(arg(C', 2), S, d, T) \wedge prefix(R.C, S.C') \wedge arg(C', 2) \neq l]]]]$

$[nf(T) \wedge L(T.\mathbf{set\_CE}(u, d, l)) \wedge T = S.\mathbf{set\_CE}(u', d, l')] \rightarrow T.\mathbf{set\_CE}(u, d, l) \equiv S.\mathbf{set\_CE}(u, d, l)$

$[nf(T) \wedge L(T.\mathbf{set\_CE}(u, d, l)) \wedge \exists C\, \exists R\, [prefix(R.C, T) \wedge callname(C) = \mathbf{set\_CE} \wedge coref(arg(C, 2), R, d, T) \wedge arg(C, 2) = l \wedge$
$\neg\exists C'\, \exists S\, [callname(C') = \mathbf{set\_CE} \wedge coref(arg(C', 2), S, d, T) \wedge prefix(R.C, S.C') \wedge arg(C', 2) \neq l]] \rightarrow T.\mathbf{set\_CE}(u, d, l) \equiv$
T

Since users other than the system security officer may be authorized to set the classification of a particular entity, we need a way of specifying which users those are. This is done in the MMS model by means of the predicate AS, defined here as follows:

$AS(v, c, r, k, T) \rightarrow [RO(v, T, sso) \vee$
$\exists S\, [some\_none\_ref_{(3,3)}(S, \mathbf{add\_AS}, \mathbf{rm\_AS}, (4, r), 2, v, 3, c, 5, k; (4, r), 2, v, 3, c, 5, k, T) \wedge \neg in\_after_1(S, \mathbf{delete\_user}, 2, v, T)]]$

That is, user v is authorized to perform procedure c at trace T with reference r as its kth argument if and only if either v is a system security officer at T or else the triple $\langle c, r, k\rangle$ has been added to v's access set at some point during the course of T without subsequently being removed.

The fourth and last of the parameter-setting procedures is **set_CCR**, which follows much the same pattern as the other three.

set_CCR

$nf(T) \rightarrow [L(T.\mathbf{set\_CCR}(u, r, x)) \leftrightarrow [logged\_in(u, T) \wedge user\_exists(u, T) \wedge container(r, T) \wedge ref\_exists(r, T) \wedge AS(u, \mathbf{set\_CCR}, r, 2, T) \wedge ref\_secure(T, \mathbf{set\_CCR}(u, r, x))]]$

$[nf(T) \wedge L(T.\mathbf{set\_CCR}(u, r, x))] \rightarrow [nf(T.\mathbf{set\_CCR}(u, r, x)) \leftrightarrow [CCR(r, T) \neq x \wedge \neg\exists S\, \exists v\, \exists y\, T = S.\mathbf{set\_CCR}(v, r, y)]$

$[nf(T) \wedge L(T.\mathbf{set\_CCR}(u, r, x)) \wedge CCR(r, T) = x] \rightarrow T.\mathbf{set\_CCR}(u, r, x) \equiv T$

$[nf(T) \wedge L(T.\mathbf{set\_CCR}(u, r, x)) \wedge T = S.\mathbf{set\_CCR}(v, r, y)] \rightarrow T.\mathbf{set\_CCR}(u, r, x) \equiv S.\mathbf{set\_CCR}(u, r, x)]$

A call to **set_CCR** preserves normal form-hood just in case (i) the container in question does not already have the CCR value to which it is about to be set, and (ii) the resulting trace does not contain back-to-back calls to **set_CCR** for the same container. If (i) fails, then the trace is equivalent to the result of deleting the last call to **set_CCR**, and if (ii) fails then the trace is equivalent to the result of deleting the penultimate call to **set_CCR**.

2.3.2 Adding and removing values from a list

In addition to the single valued security parameters discussed above, the MMS model contains security parameters in list form. These are the user's roles, role set, and access set. The values of these list parameters are set by means of pairs of procedures: one for adding values to the list, one for removing values from the list. For example, a user's role set is manipulated by the procedures **add_R** and **rm_R**:

add_R

$nf(T) \rightarrow [L(T.\mathbf{add\_R}(u, v, r)) \leftrightarrow [RO(u, T, sso) \wedge logged\_in(u, T) \wedge user\_exists(v, T) \wedge ref\_secure(T, \mathbf{add\_R}(u, v, r))]]$

$[nf(T) \wedge L(T.\mathbf{add\_R}(u, v, r))] \rightarrow [nf(T.\mathbf{add\_R}(u, v, r)) \leftrightarrow [\neg R(v, T, r) \wedge \neg\exists u'\, \exists S\, T = S.\mathbf{rm\_R}(u', v, r)]]$

$[nf(T) \wedge L(T.\mathbf{add\_R}(u, v, r)) \wedge R(v, T, r)] \rightarrow T.\mathbf{add\_R}(u, v, r) \equiv T$

$[nf(T) \wedge L(T.\mathbf{add\_R}(u, v, r)) \wedge T = S.\mathbf{rm\_R}(u', v, r)] \rightarrow T.\mathbf{add\_R}(u, v, r)) \equiv S]$

rm_R

$nf(T) \rightarrow [L(T.\mathbf{rm\_R}(u, v, r) \leftrightarrow [logged\_in(u, T) \wedge RO(u, T, sso) \wedge user\_exists(v, T) \wedge \neg RO(v, T, r) \wedge ref\_secure(T, \mathbf{rm\_R}(u, v, r))]]$

$[nf(T) \wedge L(T.\mathbf{rm\_R}(u, v, r)] \rightarrow [nf(T.\mathbf{rm\_R}(u, v, r)) \leftrightarrow [R(v, T, r) \wedge \neg\exists u'\, \exists S\, T = S.\mathbf{add\_R}(u', v, r)]]$

$[nf(T) \wedge L(T.\mathbf{rm\_R}(u, v, r) \wedge \neg R(v, T, r)] \rightarrow T.\mathbf{rm\_R}(u, v, r) \equiv T$

$[nf(T) \wedge L(T.\mathbf{rm\_R}(u, v, r) \wedge T = S.\mathbf{add\_R}(u', v, r)] \rightarrow T.\mathbf{rm\_R}(u, v, r) \equiv S$

Only a system security officer may manipulate a user's role set, and so a call to **add_R** or **rm_R** is legal only if the user issuing the call has the role of system security officer. A call to **add_R** or **rm_R** preserves normal form-hood if and only if it is nonredundant and does not cancel the effect of the procedure call that immediately precedes it.

The other two pairs of procedures that modify list-parameters follow the same pattern as **add_R** and **rm_R**.

User roles:

add_RO

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathbf{add\_RO}(u, v, r)) \leftrightarrow [[\mathrm{RO}(u, T, sso) \vee u = v] \wedge \mathrm{logged\_in}(u, T) \wedge \mathrm{R}(v, T, r) \wedge \mathrm{user\_exists}(v, T) \wedge \mathrm{ref\_secure}(T, \mathbf{add\_RO}(u, v, r))]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{add\_RO}(u, v, r))] \rightarrow [\mathrm{nf}(T.\mathbf{add\_RO}(u, v, r)) \leftrightarrow [\neg \mathrm{RO}(v, T, r) \wedge \neg \exists u' \exists S\, T = S.\mathbf{rm\_RO}(u', v, r)]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{add\_RO}(u, v, r)) \wedge \mathrm{RO}(v, T, r)] \rightarrow T.\mathbf{add\_RO}(u, v, r) \equiv T$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{add\_RO}(u, v, r)) \wedge T = S.\mathbf{rm\_RO}(u', v, r)] \rightarrow T.\mathbf{add\_RO}(u, v, r)) \equiv S]$

rm_RO

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathbf{rm\_RO}(u, v, r) \leftrightarrow [\mathrm{logged\_in}(u, T) \wedge [\mathrm{RO}(u, T, sso) \vee u = v] \wedge \mathrm{R}(v, T, r) \wedge \mathrm{user\_exists}(v, T)$
$\wedge \mathrm{ref\_secure}(T, \mathbf{rm\_RO}(u, v, r))]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{rm\_RO}(u, v, r)] \rightarrow [\mathrm{nf}(T.\mathbf{rm\_RO}(u, v, r)) \leftrightarrow [\mathrm{RO}(v, T, r) \wedge \neg \exists u' \exists S\, T = S.\mathbf{add\_R}(u', v, r)]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{rm\_RO}(u, v, r) \wedge \neg \mathrm{RO}(v, T, r)] \rightarrow T.\mathbf{rm\_RO}(u, v, r) \equiv T$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{rm\_RO}(u, v, r) \wedge T = S.\mathbf{add\_RO}(u', v, r)] \rightarrow T.\mathbf{rm\_RO}(u, v, r) \equiv S$

Access sets:

add_AS

$[\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathbf{add\_AS}(u, v, c, i, k)) \leftrightarrow$
$[\mathrm{logged\_in}(u, T) \wedge \mathrm{RO}(u, T, sso) \wedge \mathrm{ref\_exists}(i, T) \wedge [\mathrm{role}(v) \vee \mathrm{user\_exists}(v, T)] \wedge \mathrm{ref\_secure}(T, \mathbf{add\_AS}(u, v, c, i, k))]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{add\_AS}(u, v, c, i, k))] \rightarrow [\mathrm{nf}(T.\mathbf{add\_AS}(u, v, c, i, k)) \leftrightarrow$
$\forall S\, \mathrm{after\_every\_ref}_{(3,3)}(S, \mathbf{add\_AS}, \mathbf{rm\_AS}, (4, i), 2, v, 3, c, 5, k; (4, i), 2, v, 3, c, 5, k, T)$
$\wedge \neg \exists j \exists u' \exists S\, [T = S.\mathbf{rm\_AS}(u', v, c, j, k) \wedge \mathrm{coref}(i, T, j, S)]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{add\_AS}(u, v, c, i, k)) \wedge T = S.\mathbf{rm\_AS}(u', v, c, j, k) \wedge \mathrm{coref}(i, T, j, S)] \rightarrow T.\mathbf{add\_AS}(u, v, c, i, k) \equiv S]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{add\_AS}(u, v, c, i, k)) \wedge$
$\neg \forall S\, \mathrm{after\_every\_ref}_{(3,3)}(S, \mathbf{add\_AS}, \mathbf{rm\_AS}, (4, i), 2, v, 3, c, 5, k; (4, i), 2, v, 3, c, 5, k, T)] \rightarrow T.\mathbf{add\_AS}(u, v, c, i, k) \equiv T]$

rm_AS

$[\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathbf{rm\_AS}(u, v, c, i, k)) \leftrightarrow$
$[\mathrm{logged\_in}(u, T) \wedge \mathrm{RO}(u, T, sso) \wedge \mathrm{ref\_exists}(i, T) \wedge [\mathrm{role}(v) \vee \mathrm{user\_exists}(v, T)] \wedge \mathrm{ref\_secure}(T, \mathbf{rm\_AS}(u, v, c, i, k))]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{rm\_AS}(u, v, c, i, k))] \rightarrow [\mathrm{nf}(T.\mathbf{rm\_AS}(u, v, c, i, k)) \leftrightarrow$
$\exists S\, \mathrm{some\_none\_ref}_{(3,3)}(S, \mathbf{add\_AS}, \mathbf{rm\_AS}, (4, i), 2, v, 3, c, 5, k; (4, i), 2, v, 3, c, 5, k, T)$
$\wedge \neg \exists j \exists u' \exists S\, [T = S.\mathbf{add\_AS}(u', v, c, j, k) \wedge \mathrm{coref}(i, T, j, S)]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{rm\_AS}(u, v, c, i, k)) \wedge T = S.\mathbf{add\_AS}(u', v, c, j, k) \wedge \mathrm{coref}(i, T, j, S)] \rightarrow T.\mathbf{rm\_AS}(u, v, c, i, k) \equiv S]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{rm\_AS}(u, v, c, i, k)) \wedge$
$\exists S\, \mathrm{some\_none\_ref}_{(3,3)}(S, \mathbf{add\_AS}, \mathbf{rm\_AS}, (4, i), 2, v, 3, c, 5, k; (4, i), 2, v, 3, c, 5, k, T)] \rightarrow T.\mathbf{rm\_AS}(u, v, c, i, k) \equiv T]$

Predicates of the $\mathrm{some\_none\_ref}_{(m,n)}$ family allow us to say of a trace that it contains a call to a certain procedure and that this call refers to a given entity but is not followed by an occurrence of a procedure call of a certain sort referring to the same entity. More specifically, $\mathrm{some\_none\_ref}_{(m,n)}(S, c_1, c_2, (j, r), j_1, x_1, \ldots, j_m, x_m; (k, r), k_1, y_1, \ldots, k_n, y_n, T)$ states that S is a prefix of T and is followed in T by a call to procedure c_1. This call has a reference x in position j

that refers at S to the same entity that r refers to at T, and this call to c_1 has argument x_i in position j_i, for i from 1 to m, and there does not exist in T after S and the call to c_1 that follows it, any occurrence of a call to c_2 having argument y_i in position k_i, for i from 1 to n, and having in position k a reference that refers (at the trace to which the c_2 call is appended) to the same entity that r refers to at T. The predicate after_every_ref$_{(m,n)}$ is defined as the negation of some_none_ref$_{(m,n)}$.

2.3.3 Asking for values

In this section we consider the specifications of functions that a system security officer may use to ask for the roles, role set, clearance and access set of a user, the maximum classification of information that is allowed to be displayed on an output device, the classification of an entity, and the CCR value of a container. These specifications follow one of two simple patterns, the first of which being exemplified by the specification of **current_R**:

current_R

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathbf{current\_R}(u, v, r)) \leftrightarrow$
$[\mathrm{logged\_in}(u, T) \wedge [\mathrm{RO}(u, T, sso) \vee u = v] \wedge \mathrm{user\_exists}(v, T) \wedge \mathrm{ref\_secure}(T, \mathbf{current\_R}(u, v, r))]]$

$\neg\mathrm{nf}(T.\mathbf{current\_R}(u, v))$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{current\_R}(u, v, r))] \rightarrow T.\mathbf{current\_R}(u, v, r) \equiv T$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{current\_R}(u, v, r))] \rightarrow [[\mathrm{V}(T.\mathbf{current\_R}(u, v, r)) = \mathbf{True} \leftrightarrow \mathrm{R}(v, T, r)] \wedge [\mathrm{V}(T.\mathbf{current\_R}(u, v, r)) = \mathbf{False} \leftrightarrow \neg\mathrm{R}(v, T, r)]]$

Note that instead of giving the conditions under which a legal trace ending in **current_R** whose longest proper prefix is in normal form is itself in normal form, the second axiom of this specification simply states that no trace ending in **current_R** is in normal form. This implies that a legal trace ending in **current_R** whose prefix is in normal form is itself in normal form if and only if ψ, where ψ is any contradictory formula. We could just as easily have stated the second axiom this way, but the formulation that was actually used seems more to the point.

The third axiom implies that a trace ending in **current_R** is equivalent to any normal form trace that is equivalent to its longest prefix. The fourth axiom states that **current_R** returns the value *true* just in case the role set of user v at T contains the role r. The role set predicate R is defined as follows:

$\mathrm{R}(v, T, r) \leftrightarrow [[v = Root \wedge r = sso] \vee$
$\exists S\,[\mathrm{some\_none}_{(2,2)}(S, \mathbf{add\_R}, \mathbf{rm\_R}, 2, v, 3, r; 2, v, 3, r, T) \wedge \neg\mathrm{in\_after}_1(S, \mathbf{delete\_user}, 2, v, T)]]$

That is, r belongs to v's role set at T just in case there's a point in the history of T such that v has existed continuously since then and such that r was added to v's role set at that point and not thereafter removed.

Note that in the case of **current_R**, the return value of the function in question is a truth value, and the security parameter in question is a list, viz. the list of roles that some user is authorized to have. The function is used to ask whether a particular value is in the list, and returns the value *true* if it is and the value *false* if it is not. The other functions that have this character are **current_RO** and **current_AS**, which we display below. The specification of **current_AS** includes a trace definition of the predicate AS.

current_RO

$nf(T) \rightarrow [L(T.\textbf{current\_RO}(u, v, r)) \leftrightarrow [\text{logged\_in}(u, T) \wedge [RO(u, T, sso) \vee u = v] \wedge \text{user\_exists}(v, T) \wedge \text{ref\_secure}(T, \textbf{current\_RO}(u, v, r))]]$

$\neg nf(T.\textbf{current\_RO}(u, v))$

$[nf(T) \wedge L(T.\textbf{current\_RO}(u, v, r))] \rightarrow T.\textbf{current\_RO}(u, v, r) \equiv T$

$[nf(T) \wedge L(T.\textbf{current\_RO}(u, v, r))] \rightarrow$
$[[V(T.\textbf{current\_RO}(u, v, r)) = \textbf{True} \leftrightarrow RO(v, T, r)] \wedge [V(T.\textbf{current\_RO}(u, v, r)) = \textbf{False} \leftrightarrow \neg RO(v, T, r)]]$

current_AS

$nf(T) \rightarrow [L(T.\textbf{current\_AS}(u, v, c, r, k)) \leftrightarrow [\text{logged\_in}(u, T) \wedge RO(u, T, sso) \wedge \text{user\_exists}(v, T) \wedge \text{ref\_exists}(r, T) \wedge \text{ref\_secure}(T, \textbf{current\_AS}(u, v, c, r, k))]]$

$\neg nf(T.\textbf{current\_AS}(u, v, c, r, k))$

$[nf(T) \wedge L(T.\textbf{current\_AS}(u, v, c, r, k))] \rightarrow T.\textbf{current\_AS}(u, v, c, r, k) \equiv T$

$[nf(T) \wedge L(T.\textbf{current\_AS}(u, v, c, r, k))] \rightarrow [[V(T.\textbf{current\_AS}(u, v, c, r, k)) = \textbf{True} \leftrightarrow AS(v, c, r, k, T)] \wedge$
$[V(T.\textbf{current\_AS}(u, v, c, r, k)) = \textbf{False} \leftrightarrow \neg AS(v, c, r, k, T)]]$

$AS(v, c, r, k, T) \leftrightarrow [RO(v, T, sso)$
$\vee \exists S\, [\text{some\_none\_ref}_{(3,3)}(S, \textbf{add\_AS}, \textbf{rm\_AS}, (4, r), 2, v, 3, c, 5, k; (4, r), 2, v, 3, c, 5, k, T) \wedge$
$\neg \text{in\_after}_1(S, \textbf{delete\_user}, 2, v, T)]]$

In cases where the parameter in question is a single value, the function involved will simply return that value, as in the specification of the procedure **current_CU**, which returns a user's clearance and which is displayed below along with the trace definition of the MMS model predicate CU:

current_CU

$nf(T) \rightarrow [L(T.\textbf{current\_CU}(u, v, r)) \leftrightarrow [\text{logged\_in}(u, T) \wedge [RO(u, T, sso) \vee u = v] \wedge \text{user\_exists}(v, T) \wedge \text{ref\_secure}(T, \textbf{current\_CU}(u, v, r))]]$

$\neg nf(T.\textbf{current\_CU}(u, v))$

$[nf(T) \wedge L(T.\textbf{current\_CU}(u, v, r))] \rightarrow T.\textbf{current\_CU}(u, v, r) \equiv T$

$[nf(T) \wedge L(T.\textbf{current\_CU}(u, v, l))] \rightarrow [V(T.\textbf{current\_CU}(u, v)) = CU(v, T)]$

$CU(v, T) = l \leftrightarrow [\exists S\, [\text{some\_none}_{(2,1)}(S, \textbf{create\_user}, \textbf{set\_cu}, 2, v, 3, l; 2, v, T) \wedge \neg \text{in\_after}_1(S, \textbf{delete\_user}, 2, v, T)] \vee$
$[\exists S\, [\text{some\_none}(S, \textbf{set\_cu}, \textbf{set\_cu}, 2, v, 3, l; 2, v, T) \wedge \neg \text{in\_after}_1(S, \textbf{delete\_user}, 2, v, T)]]$

The other procedures whose specifications are of this second type are **current_CE**, **current_CCR**, and **current_CD**, which we display below along with the trace defintions of their associated MMS predicates CE, *CCR*, and *CD*:

current_CE

$nf(T) \rightarrow [L(T.\textbf{current\_CE}(u, r)) \leftrightarrow [\text{logged\_in}(u, T) \wedge RO(u, T, sso) \wedge \text{user\_exists}(v, T) \wedge \text{ref\_exists}(r, T) \wedge \text{ref\_secure}(T, \textbf{current\_CE}(u, r))]]$

$\neg nf(T.\textbf{current\_CE}(u, r))$

$[nf(T) \wedge L(T.\textbf{current\_CE}(u, r))] \rightarrow T.\textbf{current\_CE}(u, r) \equiv T$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{current\_CE}(u, r))] \rightarrow [\text{V}(T.\textbf{current\_CE}(u, r)) = \text{CE}(r, T)$

$\text{CE}(r, T) = l \leftrightarrow \exists S\, \exists C\, \exists s\, [[\text{callname}(C) = \textbf{set\_CE} \vee \text{callname}(C) = \textbf{cont\_create} \vee \text{callname}(C) = \textbf{obj\_create} \vee \text{callname}(C) = \textbf{downgrade}] \wedge \text{arg}(C, 2) = s \wedge \text{arg}(C, 3) = l \wedge \text{coref}(s, S, r, T) \wedge \neg\exists R\, \text{in\_after\_ref}_1(S.C.R, c, (2, r), 3, r, T) \wedge [c = \textbf{delete\_ref} \vee c = \textbf{downgrade} \vee c = \textbf{set\_CE}]$

current_CCR

$\text{nf}(T) \rightarrow [\text{L}(T.\textbf{current\_CCR}(u, r)) \leftrightarrow [\text{logged\_in}(u, T) \wedge \text{RO}(u, T, sso) \wedge \text{ref\_exists}(r, T) \wedge \text{ref\_secure}(T, \textbf{current\_CCR}(u, r))]]$

$\neg\text{nf}(T.\textbf{current\_CCR}(u, r))$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{current\_CCR}(u, r))] \rightarrow T.\textbf{current\_CCR}(u, r) \equiv T$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{current\_CCR}(u, r))] \rightarrow [\text{V}(T.\textbf{current\_CCR}(u, r)) = \text{CCR}(r, T)$

$\text{CCR}(r, T) = \textbf{True} \leftrightarrow \exists C_0\, \exists R_0\, \forall C\, \forall R\, [[\text{callname}(C) = \textbf{set\_CCR} \wedge \text{prefix}(R.C, T) \wedge \text{coref}(\text{arg}(C, 2), R, r, T)] \leftrightarrow \text{prefix}(R.C, R_0.C_0)] \wedge \text{arg}(C, 3) = \textbf{True}]$

$\text{CCR}(r, T) = \textbf{False} \leftrightarrow \text{CCR}(r, T) \neq \textbf{True}$

current_CD

$\text{nf}(T) \rightarrow [\text{L}(T.\textbf{current\_CD}(u, d)) \leftrightarrow [\text{logged\_in}(u, T) \wedge \text{RO}(u, T, sso) \wedge \text{device}(d)]]$

$\neg\text{nf}(T.\textbf{current\_CD}(u, d))$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{current\_CD}(u, d))] \rightarrow T.\textbf{current\_CD}(u, d) \equiv T$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{current\_CD}(u, d))] \rightarrow \text{V}(T.\textbf{current\_CD}(u, d)) = \text{CD}(d, T)]$

$\text{CD}(d, T) = x \leftrightarrow [\exists S\, \exists C\, [\text{callname}(C) = \textbf{set\_CD} \wedge x = \text{arg}(C, 3) \wedge \forall D\, \forall R\, [\text{prefix}(R.D, S.C) \leftrightarrow [\text{prefix}(R.D, T) \wedge \text{callname}(D) = \textbf{set\_CD} \wedge \text{coref}(\text{arg}(D, 2), R, d, T)]]] \vee [x = \text{unclassified} \wedge \neg\exists S\, \exists C\, [\text{callname}(C) = \textbf{set\_CD} \wedge \text{prefix}(S.C, T) \wedge \text{coref}(\text{arg}(C, 2), S, d, T)]]$

2.4 References and Entities

In the MMS security model, references to entities can be *direct* or *indirect*. A direct reference is a number that serves as the name of an entity in the message system. An indirect reference is a sequence $n_1 : \ldots : n_{i+1}$ of numbers, where $n_1 : \ldots : n_i : n_{i+1}$ represents the n_{i+1}th element in the container to which $n_1 : \ldots : n_i$ refers. In the MMS model trace specification, references are defined recursively by the following axioms, where k is a variable that ranges over the positive integers.

Recursive axioms

$\text{direct\_ref}(k) \wedge \forall r\, \exists k[\text{direct\_ref}(r) \rightarrow r = k]$
$\text{direct\_ref}(r) \rightarrow \text{reference}(r)$
$\text{reference}(r) \rightarrow \text{reference}(r : k)$
$\text{reference}(r) \rightarrow [\text{direct\_ref}(r) \vee \exists s\, \exists k\ [\text{reference}(s) \wedge r = s : k]]$

2.4.1 Identifying references

It can safely be assumed that in the course of writing and modifying messages, entities will frequently change their composition. Entities can be inserted in one another, deleted from one another, or deleted altogether, for example. Yet we want to ensure that an entity retains its security classification through such changes, unless the classification is explicitly modified, and we want the same access privileges to apply to an entity after it is modified, unless these access privileges are modified. In order to use references to identify an entity after it has been changed we define the conditions under which one reference can be said to refer to the *same entity* at trace T that another reference refers to at trace S. This is done by means of the coref predicate, a four-place relation that is governed by the following axioms:

Coreference

$\text{coref}(r, T, r, T)$

$\text{coref}(r, T, s, R) \rightarrow \text{coref}(s, R, r, T)$

$\text{coref}(r, T, s, R) \rightarrow [\text{coref}(s, R, t, S) \rightarrow \text{coref}(r, T, t, S)]$

$\text{callname}(C) = \textbf{insert} \rightarrow$
$[\text{coref}(\text{arg}(C, 2), T, \text{arg}(C, 3) : \text{arg}(C, 4), T.C \wedge [0 < k < \text{arg}(C, 4) \rightarrow \text{coref}(\text{arg}(C, 3) : k, T, \text{arg}(C, 3) : k, T.C)] \wedge [[k > \text{arg}(C, 4) \wedge \text{ref\_exists}(\text{arg}(C, 3) : k, T)] \rightarrow \text{coref}(\text{arg}(C, 3) : k, T, \text{arg}(C, 3) : (k + 1), T.C)]$

$[[\text{callname}(C) = \textbf{delete\_ref} \wedge \text{coref}(\text{arg}(C, 2), S, s : k, T)] \vee [\text{callname}(C) = \textbf{remove} \wedge \text{coref}(\text{arg}(C, 3), S, \text{arg}(C, 2) : \text{arg}(C, 4), T) \wedge k = \text{arg}(C, 4) \wedge s = \text{arg}(C, 3)] \rightarrow [0 < n < k \rightarrow \text{coref}(s : n, T, s : n, T.C)] \wedge [[n > k \wedge \text{ref\_exists}(s : n, T)] \rightarrow \text{coref}(s : n, T, s : (n - 1), T.C)]]$

$[\text{callname}(C) \neq \textbf{delete\_ref} \wedge \text{callname}(C) \neq \textbf{insert} \wedge \text{callname}(C) \neq \textbf{remove}] \rightarrow [\text{coref}(r, T, t, S) \rightarrow \text{coref}(r, T, t, S.C)]$

$\text{coref}(r, T, s : k, T) \rightarrow \exists C \exists S\, [\text{prefix}(S.C, T) \wedge \text{callname}(C) = \textbf{insert} \wedge \text{coref}(\text{arg}(C, 2), S, r, T) \wedge \text{coref}(\text{arg}(C, 3), S, s, T)]$

The first three axioms state that coref is an equivalence relation if regarded it as a binary relation on pairs $\langle r, T \rangle$, where r is a reference and T is a trace. The third, fourth, and fifth axioms describe how the identity of an entity is preserved when modified using the procedures **insert**, **delete_ref**, and **remove**, which are the only entity-modifying procedures presented here. If a container is modified by **insert**ing an entity into it at position k, then the entities at positions 1 through $k - 1$ in this container can still be referred to in the same way; of course, the newly inserted entity becomes the kth element of the container; and, finally, those entities formerly in positions $k + 1$ or higher now have their positions incremented by one. If, on the other hand, a container is modified by **delete_ref**ing or **remove**ing an entity from position k, then the entities that were in positions 1 to $k - 1$ keep the same position, whereas entities in positions $k + 1$ and higher have their positions decremented by one. Coreference is not modified by procedures other than **delete_ref**, **insert**, and **remove**. Finally, coreference holds between references r and $s : k$ only if r has been inserted into s.

Most of what needs to be said about entities can be put entirely in terms of references, but for some purposes, such as the definition of potential modification, we will need to say things directly about entities. Entities occur in this specification as the value of an entity function E which must satisfy these conditions:

$\forall r \forall s \forall T \forall R\, [\text{coref}(r, T, s, R) \rightarrow E(r, T) = E(s, R)]$

$\forall r \forall T\, [\text{ref\_exists}(r, T) \rightarrow \exists y\, y = E(r, T)$

$\text{entity\_exists}(x, T) \leftrightarrow \exists r\, x = E(r, T)$

$\text{entity}(x) \leftrightarrow \exists T\, \text{entity\_exists}(x, T)$

$[\text{entity}(x) \wedge \text{entity}(y) \rightarrow [x = y \leftrightarrow \exists S \exists T \exists r_1 \exists r_2\, [E(r_1, S) = x \wedge E(r_2, T) = y \wedge \text{coref}(r_1, S, r_2, T)]]$

$\forall x \forall T\, [\text{entity\_exists}(x, T) \rightarrow \exists z\, [z = \text{value}(x, T) \wedge \text{string}(z)]]$

The based_on relation, which identifies those references that refer to the pieces of an entity, is defined recursively:

$\forall r \forall k \forall e\, [\text{ref\_exists}(r : k, T) \wedge E(r, T) = e] \rightarrow \text{based\_on}(r : k, e, T)$

$\forall r \forall k \forall e\, [\text{ref\_exists}(r : k, T) \wedge \text{based\_on}(r, e, T)] \rightarrow \text{based\_on}(r : k, e, T)$

2.4.2 Creating, modifying, and displaying references

The procedures **obj_create**, **cont_create**, **insert**, **remove**, **delete_ref**, *identify*, and **display** can be used to create, modify, and display references.

Entities are of two kinds: objects and containers. An *object* is an atomic or text-bearing entity; a *container* is any sequence of entities. Objects and containers are created using the procedures **obj_create** and **cont_create**, respectively.

obj_create

$\text{nf}(T) \rightarrow [\text{L}(T.\textbf{obj\_create}(u, k, l, p)) \leftrightarrow [\text{logged\_in}(u, T) \wedge \text{string}(p) \wedge \neg\exists S\, [\text{prefix}(S, T) \wedge \text{ref\_exists}(k, S)] \wedge \text{ref\_secure}(T, \textbf{obj\_create}(u, k, l, p))]]$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{obj\_create}(u, k, l, p))] \rightarrow \text{nf}(T.\textbf{obj\_create}(u, k, l, p))$

cont_create

$\text{nf}(T) \rightarrow [\text{L}(T.\textbf{cont\_create}(u, k, l)) \leftrightarrow [\text{logged\_in}(u, T) \wedge \neg\exists S\, [\text{prefix}(S, T) \wedge \text{ref\_exists}(k, T)] \wedge \text{ref\_secure}(T, \textbf{cont\_create}(u, k, l))]]$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{cont\_create}(u, k, l))] \rightarrow \text{nf}(T.\textbf{cont\_create}(u, k, l))$

When an object is created, it is assigned a direct reference k, a security classification l, and a value p, which is the textual string associated with the object. When a container is created it is assigned only a direct reference and a security classification. Entities are inserted into and removed from containers using the **insert** and **remove** commands, respectively.

insert

$\text{nf}(T) \rightarrow [\text{L}(T.\textbf{insert}(u, x, y, k)) \leftrightarrow [\text{logged\_in}(u, T) \wedge [\text{logged\_in}(u, y, T) \rightarrow \text{CU}(u, T) \geq \text{CE}(y, T)] \wedge \text{ref\_exists}(x, T) \wedge \text{ref\_exists}(y, T) \wedge \text{CE}(x, T) \leq \text{CE}(y, T) \wedge \neg\text{part\_of}(y, x, T) \wedge k \geq 0 \wedge \text{AS}(u, \textbf{insert}, x, 2, T) \wedge \text{container}(y) \wedge \text{AS}(u, \textbf{insert}, y, 3, T) \wedge \text{ref\_secure}(T, \textbf{insert}(u, x, y, k)) \wedge [k = 1 \vee \exists s\, \text{H}(s, T, y, k - 1)]]]$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{insert}(u, x, y, k))] \rightarrow \text{nf}(T.\textbf{insert}(u, x, y, k))$

remove

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathbf{remove}(u, x, y, k)) \leftarrow [\mathrm{logged\_in}(u, T) \wedge \mathrm{ref\_exists}(x, T) \wedge k \geq 0 \wedge \mathrm{AS}(u, \mathbf{remove}, x, 2, T) \wedge \mathrm{AS}(u, \mathbf{remove}, y, 3, T) \wedge \mathrm{coref}(y, T, x : k, T) \wedge \mathrm{ref\_secure}(T, \mathbf{remove}(u, x, y, k))]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{remove}(u, x, y, k))] \rightarrow \mathrm{nf}(T.\mathbf{remove}(u, x, y, k))$

Note that **insert**ing and **remove**ing always preserve normal form-hood; this is intuitively correct, since one wants any legal modification of an entity to constitute a change in the state of the module. Note also that legality requires the classification of the inserted entity x to be no greater than the classification of the container y into which it is inserted. This requirement is part of the MMS definition of state security.

removeing an entity from a container does not destroy the entity being removed. For this the **delete_ref** command must be used.

delete_ref

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathbf{delete\_ref}(u, r)) \leftarrow [\mathrm{logged\_in}(u, T) \wedge \mathrm{ref\_exists}(r, T) \wedge \mathrm{AS}(u, \mathbf{delete\_ref}, r, 2, T) \wedge \mathrm{ref\_secure}(T, \mathbf{delete\_ref}(u, r))]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{delete\_ref}(u, r))] \rightarrow \mathrm{nf}(T.\mathbf{delete\_ref}(u, r))$

Note that **delete_ref**ing a reference, like **remove**ing a reference, preserves normal form-hood.

We have noted that **delete_ref** can, whereas **remove** cannot, be used to destroy, *i.e.* truly delete, an entity. The reader may wonder why this is so, since nothing about the way the two procedures are specified suggests this. The answer lies in the definition of existence for references:

$\mathrm{ref\_exists}(r, T) \leftarrow \exists S\, \exists C\, [\mathrm{some\_none\_ref}_{(0,0)}(S, c, \mathbf{delete\_ref}, (2, r); (2, r), T) \wedge [c = \mathbf{obj\_create} \vee c = \mathbf{cont\_create}]]$

An entity (a reference, from the user's point of view) exists just in case there is a point at which it was created and after which it was never deleted.

An entity can be displayed to an output device in any of a number of ways. One can display its value, its classification, its CCR value, its message type, or its direct reference. Hence the range of legal switches in the third argument of the **display** procedure. Note, however, that **display** is a procedure, not a function. Since **display**ed material need not be returned to the console that executed it, treating **display** as a function would not be appropriate.

display

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathbf{display}(u, r, f, d)) \rightarrow [\mathrm{logged\_in}(u, T) \wedge \mathrm{ref\_exists}(r, T) \wedge \mathrm{device}(d) \wedge \mathrm{CU}(u, T) \geq \mathrm{CE}(r, T) \wedge \mathrm{CD}(d, T) \geq \mathrm{CE}(r, T) \wedge \mathrm{AS}(u, \mathbf{display}, r, 2, T) \wedge [f = \mathit{value} \vee f = \mathit{direct\_ref} \vee f = \mathit{classification} \vee f = \mathit{type} \vee f = \mathit{ccr\_value}] \wedge [f = \mathit{value} \rightarrow [\mathrm{part\_of}(r, d, T) \wedge \exists S\, T = S.\mathbf{display}(u, r, \mathit{classification}, d)]] \wedge \forall z\, [[f = \mathit{direct\_ref} \wedge \mathrm{based\_on}(r, z, T) \wedge \mathrm{CCR}(z, T) = \mathbf{True}] \rightarrow \mathrm{CU}(u, T) \geq \mathrm{CE}(z, T)]] \wedge \mathrm{ref\_secure}(T, \mathbf{display}(u, r, f, d))]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{display}(u, r, f, d))] \rightarrow [\mathrm{nf}(T.\mathbf{display}(u, r, f, d)) \leftarrow \forall S\, \forall C\, [[\mathrm{prefix}(S.C, T) \wedge \mathrm{coref}(\mathrm{arg}(C, 2), S, r, T) \wedge \mathrm{callname}(C) = \mathbf{display} \wedge \mathrm{arg}(C, 3) = f \wedge \mathrm{arg}(C, 4) = d] \rightarrow \exists R\, \exists D\, [\mathrm{prefix}(S, R) \wedge [[\mathrm{callname}(D) = \mathbf{display} \wedge \mathrm{arg}(D, 4) = d \wedge \mathrm{coref}(\mathrm{arg}(D, 2), R, r, T) \wedge \mathrm{arg}(D, 3) \neq f] \vee [\mathrm{callname}(D) = \mathbf{delete\_ref} \wedge \mathrm{coref}(\mathrm{arg}(D, 2), R, r, T)] \vee [\mathrm{arg}(D, 4) = d \wedge \mathrm{callname}(D) = \mathbf{identify}]]]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathbf{display}(u, r, f, d)) \wedge \exists S\, \exists C\, [\mathrm{prefix}(S.C, T) \wedge \mathrm{coref}(\mathrm{arg}(C, 2), S, r, T) \wedge \mathrm{callname}(C) = \mathbf{display} \wedge \mathrm{arg}(C, 3) = f \wedge \mathrm{arg}(C, 4) = d \wedge \neg \exists R\, \exists D\, [\mathrm{prefix}(S, R) \wedge [[\mathrm{callname}(D) = \mathbf{display} \wedge \mathrm{arg}(D, 4) = d \wedge \mathrm{coref}(\mathrm{arg}(D, 2), R, r, T) \wedge$

$\text{arg}(D, 3) \neq f] \vee [\text{callname}(D) = \textbf{delete\_ref} \wedge \text{coref}(\text{arg}(D, 2), R, r, T)] \vee [\text{arg}(D, 4) = d \wedge \text{callname}(D) = \textbf{identify}]]]] \rightarrow T.\textbf{display}(u, r, f, d) \equiv T$

The special conditions in the legality definition associated with the cases where $f = value$ and $f = direct\_ref$ help to ensure that the MMS properties of State Security and CCR Security hold. The normal form statement says that a call to **display** preserves normal form-hood only if it is not made redundant by previous calls to **display**.

Next we come to the procedure **identify**, which allows one to display a user's user_id,[5] clearance, access set, roles, and role set on an output device. **identify**, like **display** is a procedure rather than a function and for the very same reason.

identify

$\text{nf}(T) \rightarrow [\text{L}(T.\textbf{identify}(u, N, f, d)) \leftrightarrow [\text{logged\_in}(u, T) \wedge \text{device}(d) \wedge \text{user}(N) \wedge [f = user\_id \vee f = \text{clearance} \vee f = access\_set \vee f = \text{role\_set} \vee f = \text{roles}] \wedge \text{ref\_secure}(T, \textbf{identify}(u, N, f, d))]]$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{identify}(u, N, f, d))] \rightarrow [\text{nf}(T.\textbf{identify}(u, N, f, d)) \leftrightarrow \forall S \forall C [[\text{prefix}(S.C, T) \wedge \text{callname}(C) = \textbf{identify} \wedge \text{arg}(C, 2) = N \wedge \text{arg}(C, 3) = f \wedge \text{arg}(C, 4) = d] \rightarrow \exists R \exists D [\text{prefix}(S, R) \wedge \text{arg}(D, 4) = d \wedge [[\text{callname}(D) = \textbf{identify} \wedge \text{arg}(D, 2) = N \wedge \text{arg}(D, 3) \neq f] \vee \text{callname}(D) = \textbf{display}]]$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{identify}(u, N, f, d)) \wedge \exists S \exists C [\text{prefix}(S.C, T) \wedge \text{callname}(C) = \textbf{identify} \wedge \text{arg}(C, 2) = N \wedge \text{arg}(C, 3) \neq f \wedge \text{arg}(C, 4) = d \wedge \neg \exists R \exists D [\text{prefix}(S.C, R) \wedge \text{arg}(D, 4) = d \wedge [[\text{arg}(D, 2) = N \wedge \text{callname}(D) = \textbf{identify} \wedge \text{arg}(D, 3) = f] \vee \text{callname}(D) = \textbf{display}]]] \rightarrow T.\textbf{identify}(u, N, f, d) \equiv T$

2.5 Downgrading and Releasing

The final two procedures in the specification are **downgrade** and **release**. Since these procedures always modify the state of the module, they are always normal form preserving.

downgrade

$\text{nf}(T) \rightarrow [\text{L}(T.\textbf{downgrade}(u, r, l) \leftrightarrow [\text{RO}(u, downgrader, T) \wedge \text{logged\_in}(u, T) \wedge \text{user\_exists}(u, T) \wedge \text{ref\_exists}(r, T) \wedge \text{CE}(r, T) > l \wedge \forall t \forall k [\text{H}(t, T, r, k) \rightarrow \text{CE}(t, T) \leq l] \wedge \text{callname}(C) = \textbf{downgrade} \wedge \text{AS}(u, \textbf{downgrade}, r, 2, T) \wedge \text{ref\_secure}(T, \textbf{downgrade}(u, r, l))]]$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{downgrade}(u, r, l)] \rightarrow \text{nf}(T.\textbf{downgrade}(u, r, l))$

release

$\text{nf}(T) \rightarrow [\text{L}(T.\textbf{release}(u, r) \leftrightarrow [\text{RO}(u, releaser, T) \wedge \text{user\_exists}(u, T) \wedge \text{logged\_in}(u, T) \wedge \text{ref\_exists}(r, T) \wedge \text{T}(r, T) = DM \wedge \text{AS}(u, \textbf{release}, r, 2, T) \wedge \text{ref\_secure}(T, \textbf{release}(u, r))]]$

$[\text{nf}(T) \wedge \text{L}(T.\textbf{release}(u, r)] \rightarrow \text{nf}(T.\textbf{release}(u, r))$

Note that the legality condition of **downgrade** requires the issuer of the command to have the role of *downgrader*. Similarly an issuer of **release** must have the role of *releaser*. These requirements ensure that the MMS conditions of Downgrade Security and Release Security are met. Also, in order to ensure State Security, a container must not be downgraded to have a classification lower than that of any of the entities that it contains.

[5] Displaying a user_id is a trivial operation, since in this specification we do not distinguish user names from user_ids.

2.6 Security

The MMS model prescribes eight basic security properties: *state* security, *access* security, *copy* security, *CCR* security, *translation security*, *set* security, *downgrade* security, and *release* security. In this section we consider, one by one, each of these security properties and give the outline of a proof that they are satisfied by any module that meets the specification outlined above.

2.6.1 State Security

The original MMS definition of state security has five parts: (i) no container may have a lower clearance than the entities it contains, (ii) no entity displayed on a user's terminal may have a classification that is higher than the user's clearance, (iii) no entity may be displayed on an output device unless its classification is displayed along with it, (iv) no user may have a role that does not belong to his current role set, and (v) the actual classification of an output device may not exceed its authorized maximum classification. These requirements are specified in the following definition of the trace predicate state_secure.

$$\text{state\_secure}(T) \leftrightarrow [\forall r \forall s \forall k\, [\mathrm{H}(r, T, s, k) \rightarrow \mathrm{CE}(s, T) \leq \mathrm{CE}(r, T)] \wedge \forall o \forall u \forall k \forall r\, [[\mathrm{H}(o, T, r, k) \wedge \text{logged\_in}(u, o, T)] \rightarrow \mathrm{CU}(u, T) \geq \mathrm{CE}(r, T)] \wedge \forall o \forall x\, [\mathrm{D}(o, x, \textit{value}, T) \rightarrow \mathrm{D}(o, x, \textit{classification}, T)] \wedge \forall u\, [\mathrm{RO}(u, T, r) \rightarrow \mathrm{R}(u, T, r)] \wedge \forall o\, [\mathrm{CD}(o, T) \geq \mathrm{CE}(o, T)]]$$

Proving that state security holds simply means proving that every legal trace satisfies this predicate, *i.e.*

Lemma 1 (State Security) $\forall T\, [\, \mathrm{L}(T) \rightarrow \text{state\_secure}(T)\,]$.

Proof: Consider the first clause. Suppose that $\mathrm{L}(T)$ and $\mathrm{H}(r, T, s, k)$ hold. Then, by the definition of H, it follows that $r : k$ is coreferential with s. By the axioms for coref, this means that the entity $E(s, T)$, to which s refers at T, was at some time inserted into the entity $E(r, T)$. Since $\mathrm{L}(T)$ holds, every prefix of T must be legal, and so the insertion of $E(s, T)$ into $E(r, T)$ must have been a legal operation. Hence, at the time of insertion S, it must have been true that $\mathrm{CE}(s, S) \leq \mathrm{CE}(r, S)$. But since T is a legal trace, all of its prefixes are legal, and since the downgrade operation is specified so that a container cannot legally be downgraded to below the level of its most highly classified element, and in view of the Downgrade Security Lemma (proved below), it follows that $\mathrm{CE}(s, T) \leq \mathrm{CE}(r, T)$.

Consider the second clause, and suppose that $\mathrm{L}(T)$, $\mathrm{H}(o, T, r, k)$ and $\text{logged\_in}(u, o, T)$ all hold. From $\mathrm{H}(o, T, r, k)$ it follows that $\text{coref}(o : k, T, r, T)$, and since $\text{logged\_in}(u, o, T)$, it must be that u inserted $E(r, T)$ into o during his current session. (The latter follows from the last coreference axiom and from the fact that logging out is only legal if one's terminal is clear, so that o must have been empty when u logged in.) Since u himself or herself inserted $E(r, T)$ into o, it follows from the legality condition of **insert** that u had the necessary clearance at the time of insertion. Since references can only be downgraded, and since the clearance of a user cannot be changed except when he is logged out, it follows that $\mathrm{CU}(u, T) \geq \mathrm{CE}(r, T)$.

Regarding the third clause, suppose that $\mathrm{D}(o, x, \textit{value}, T)$ holds. Then, by the definition of D, it follows that from some time in the history of T, say at trace S, the value of x was displayed on o. The legality clause for **display** implies that the last procedure call in S is the operation of

displaying x's classification on o. This classification must still be displayed at T, since x has not been removed from o in the meantime, hence $D(o, x, classification, T)$ holds as required.

Regarding the fourth clause, suppose that $RO(u, T, r)$ holds. Without loss of generality, suppose that u is not *Root*, T is not the empty trace, and r is not *sso*. Then, by the definition of RO, it follows that r was, at some time S in the history of T, added but never thereafter removed from the list of u's roles. But then by the legality condition of **add_R** it follows that r was in u's role set at S. And, since R has been a role of u continuously since S, it follows from the legality condition of **rm_R** that r was never removed from u's role set. Hence $R(u, T, r)$ as required.

Regarding the fifth clause, suppose, without loss of generality, that $CD(o, T)$ is higher than the value *unclassified*. Then it was set higher at some point in the history of T; let S be the moment of the last such setting, and let l be the level to which CD was last set. The legality condition of **set_CD** implies that $l \geq CE(o, S)$ holds. Now consider moments since S; by hypothesis, CD has not been modified since S, but if CE has been modified, then, by the legality condition of **set_CE**, it follows that any such modification was to a level no greater than l. Hence $CD(o, T) \geq CE(o, T)$, as required. ∎

2 6.2 Access Security

The original MMS model definition of access security states that a user may perform a given operation involving an entity only if the triple consisting of (a) either the user or one of his roles, (b) the operation, and (c) the entity's position appear in the entity's access set. In order to state this requirement in terms of traces we define the predicate acc_secure, which holds of a trace and a procedure call just in case the given procedure call itself does not violate access security.

$$acc\_secure(T, C) \leftarrow \forall k\,[reference(arg(C, k)) \rightarrow [AS(arg(C, 1), callname(C), arg(C, k), k, T) \vee \exists r\,[RO(arg(C, 1), T, r) \wedge AS(r, callname(C), arg(C, k), k, T)]]]$$

Proving that the module specified here meets the requirement of access security means proving that every legal occurrence of a procedure call satisfies this predicate, *i.e.* we must prove:

Lemma 2 (Access Security) $\forall T \forall C\,[\,L(T.C) \rightarrow acc\_secure(T, C)\,]$.

Proof: By the definition of ref_secure and by inspection of the legality condition of each procedure. ∎

2.6.3 Set Security

Set security is really two requirements: (i) a change in the maximum classification of a device or the classification or role set of a user may only be made by a user with the role of system security officer, and (ii) a change in the role of a user may only be made by the user himself or by a system security officer. Proving that our specification meets this requirement means proving that every legal occurrence of a procedure call satisfies the predicate set_secure:

$$set\_secure(T, C) \leftarrow [\forall o \forall x\,[[CD(o, T) \neq CD(o, T.C) \vee CU(x, T) \neq CU(x, T.C) \vee \exists l \neg[R(x, T, l) \equiv R(x, T.C, l)]] \rightarrow RO(arg(C, 1), T, sso)] \wedge [\forall u\,[\exists l \neg[RO(u, T, l) \equiv RO(u, T.C, l)] \rightarrow [u = arg(C, 1) \vee RO(u, T, sso)]]]$$

Lemma 3 (Set Security) $\forall T \forall C [\, \mathrm{L}(T.C) \rightarrow \mathrm{set\_secure}(T, C)\,]$.

Proof: There are four cases to consider. To begin with, consider the case where $\mathrm{CD}(o, T) \neq \mathrm{CD}(o, T.C)$ holds. From this and the definition of CD it follows that callname(C) = **set_CD**. Since L($T.C$) holds, we have by the legality axiom for **set_CD** that RO(arg(C, 1), T, *sso*), as required.

Suppose next that $\mathrm{CU}(x, T) \neq CU(x, T.C)$. From this and the definition of CU it follows that callname(C) = **set_cu**. Since, again, L($T.C$) holds, we have by the legality axiom for **set_cu** that RO(arg(C, 1), T, *sso*), as required.

Suppose that $\neg[\mathrm{R}(x, T, l) \equiv \mathrm{R}(x, T.C, l)]$ holds; there are two cases. If $\mathrm{R}(x, T, l)$ holds but $\mathrm{R}(x, T.C, l)$ does not, then callname(C) = **rm_R**, by the definition of R. But since L($T.C$) holds, it follows from the legality axiom for **rm_R** that RO(arg(C, 1), T, *sso*), as required. If $\mathrm{R}(x, T.C, l)$ holds but $\mathrm{R}(x, T, l)$ does not, then callname(C) = **add_R**, by the definition of R. By a similar argument, it follows again that RO(arg(C, 1), T, *sso*).

Finally, suppose that $\neg[\mathrm{RO}(u, T, l) \equiv \mathrm{RO}(u, T.C, l)]$; the argument and case structure are exactly as for the previous case, except that in this case it follows that $[u = \mathrm{arg}(C, 1) \vee \mathrm{RO}(u, T, sso)]$. ∎

2.6.4 Downgrade Security

To express downgrade security we need the following predicate, which holds of an occurrence of a procedure call just in case that occurrence either brings about no downgrade in the classification of any entity, or, if it does, then it is an invocation of the **downgrade** procedure by a user with the role of *downgrader*.

$\mathrm{downgrade\_secure}(T, C) \leftrightarrow \forall x\,[[\neg \mathrm{device}(x) \wedge \mathrm{CE}(x, T) > \mathrm{CE}(x, T.C)] \rightarrow$
[callname(C) = **downgrade**$\wedge$
RO(arg(C, 1), T, *downgrader*)]]

To establish Downgrade Security we prove:

Lemma 4 (Downgrade Security) $\forall T \forall C [\, \mathrm{L}(T.C) \rightarrow \mathrm{downgrade\_secure}(T, C)\,]$.

Proof: Suppose that $\mathrm{CE}(x, T) > \mathrm{CE}(x, T.C)$ holds. Since x has a classification at T, C must not be a call to either the procedures **cont_create** or **obj_create**, and since x is not a device, C must not be a call to **set_CE**, either. Hence, by the definition of CE it must be a call to **downgrade**, as required. And since $T.C$ is legal, it follows by the legality axiom of **downgrade** that RO(arg(C, 1), T, *downgrader*), as required. ∎

2.6.5 Release Security

For establishing release security we need this predicate.

$\mathrm{release\_secure}(T, C) \leftrightarrow [[\mathrm{T}(r, T) = RM \rightarrow [\mathrm{T}(r, T.C) = RM \wedge \forall u\,[\mathrm{RE}(r, T, u) \rightarrow \mathrm{RE}(r, T.C, u)]] \wedge [[\mathrm{T}(r, T) \neq RM \wedge \mathrm{T}(r, T.C) = RM] \rightarrow [\mathrm{RE}(r, T.C, \mathrm{arg}(C, 1)) \wedge \mathrm{callname}(C) = \mathbf{release} \wedge \mathrm{arg}(C, 2) = r \wedge \mathrm{RO}(\mathrm{arg}(C, 1), T, releaser) \wedge \mathrm{T}(r, T) = DM]]]$

Lemma 5 (Release Security) $\forall T \forall C$ [L($T.C$) $\rightarrow$ release_secure(T, C)].

Proof: There are two cases. Suppose that T(r, T) = *RM*; then callname(C) $\neq$ **release**, by the legality axiom for **release**, which also implies that RE(r, T, u) $\rightarrow$ RE($r, T.C, u$), since T is a prefix of $T.C$.

Suppose instead that T(r, T) $\neq$ *RM* but T(r,T.C) = *RM*. In this case it follows from the definition of T that callname(C) = **release** and arg(C, 2) = r, as required. By the legality of $T.C$ and by the legality axiom for **release** it follows that RO(arg(C, 1), T, *releaser*) holds, as required, and by the definition of *DM* it follows that T(r, T) = *DM*, as required. ∎

2.6.6 Translation Security

In order to prove translation security we need this predicate:

trans_secure(T, C) $\leftrightarrow$ $\forall o \forall x \forall z \forall u$ [[logged_in($u, o, T.C$) $\wedge$ direct_ref(x)$\wedge$
D(o, x, *direct_ref*, $T.C$)] $\rightarrow$ $\forall s$ [$\exists r \exists k$ [arg(C, k) = $r \wedge E(r, T) = E(x, T) \wedge$ based_on(r, z, T) $\wedge E(s, T) = z \wedge$ CCR(s, T)] $\rightarrow$ CU(u, T) $\geq$ CE(s, T)]]]

Lemma 6 (Translation Security) $\forall T \forall C$ [L($T.C$) $\rightarrow$ trans_secure(T, C)].

Proof: Suppose, as required, that a direct reference to an entity is being displayed on the terminal of a user who is logged in; i.e. suppose that D(o, x, *direct_ref*, $T.C$) and the other hypotheses of trans_secure(T, C) hold. Then, by the definition of D, there is a prefix $S.C$ of T such that C is a call to **display** and *direct_ref* is its third argument. By the legality of $S.C$ and the legality axiom of **display**, it follows that CU(u, T) $\geq$ CE(s, T), as required. ∎

2.6.7 Copy and CCR Security

Copy and CCR security are based on the notion of *potential modification*, for which a rather complicated trace definition is given in the appendix to this report. The idea behind potential modification, is this: a procedure call C potentially modifies a reference r at a trace T with an entity y as a contributing factor if and only if there is at least one trace S that *would* be equivalent to T except for the values of some entities,[6] such that C executed at S modifies some function F of r, where F is taken from the following list: access set, containment, value, CCR value, classification, or message type, and where the F-value of r varies depending on the (entity) value of y. Copy security requires that any such contributing factor have a classification no higher than r itself has. CCR security requires that if y is an entity that is based on a CCR container x, and if y is a contributing factor in the potential modification of some reference, then the security clearance of the user submitting the procedure call must be at least as great as the classification of x.

The relevant trace predicates for copy and CCR security are these:

copy_secure(T, C) $\leftrightarrow$ $\forall r \forall y$ [p_modify(C, r, T, y) $\rightarrow$ $\exists x$ [$y = E(x, T) \wedge$ CE(x, T) $\leq$ CE(r, T)]

[6] Such an S would *be* trace equivalent to T, since entity values do not figure in trace legality and since there are no functions in the module that return them as values. Equivalence up to a single reference is expressed in the o_equiv predicate.

$\text{CCR\_secure}(T, C) \leftrightarrow \forall u \forall r \forall z \forall y [[\exists k\, r = \text{arg}(C, k) \wedge \text{based\_on}(r, y, T) \wedge \exists s [y = E(s, T) \wedge \text{CCR}(s, T) \wedge \text{p\_modify}(C, z, T, E(r, T))] \rightarrow \text{CU}(u, T) \geq \text{CE}(z, T)]$

Lemma 7 (Reference Security) $\forall T \forall C [\, \text{L}(T.C) \rightarrow [\text{copy\_secure}(T, C) \wedge \text{CCR\_secure}(T, C)\,]$.

Proof: Suppose that $\text{L}(T.C)$ holds. Since $\text{ref\_secure}(T, C)$ holds of each T and C such that $T.C$ is legal, it suffices to prove that $\text{p\_modify}(C, r, T, y) \rightarrow \exists s \exists k [\text{coref}(r, T, s, T) \wedge \text{arg}(C, k) = s\,]$.

Assume that $\text{p\_modify}(C, r, T, y)$ holds. Let S be given such that $S \equiv T$ and $\neg \text{e\_equiv}(S.S.C, E(r, S))$ both hold. Let F be an entity function for which S and $S.C$ are not equivalent. Note that since $E(r, S)$ is assumed to be well-defined, it follows that C is not **cont_create** or **obj_create**. Note also that no matter which entity function F is, it follows from the definition of the F-predicate (e.g. the definition of $\text{CE}(x, T)$) that C must be a certain procedure that sets the value of F. Obviously, some reference to $E(r, T)$ will have to be mentioned in the call to any procedure that sets the value of F for $E(r, S)$, and so it follows that $\exists s \exists k [\text{coref}(r, T, s, T) \wedge \text{arg}(C, k) = s$, as required. ∎

Finally, we summarize the eight basic security properties in the predicate MMS_secure:

$\text{MMS\_secure}(T, C) \leftrightarrow [\text{acc\_secure}(T, C) \wedge \text{copy\_secure}(T, C) \wedge \text{CCR\_secure}(T, C) \wedge \text{trans\_secure}(T, C) \wedge \text{set\_secure}(T, C) \wedge \text{downgrade\_secure}(T, C) \wedge \text{release\_secure}(T, C) \wedge \text{state\_secure}(T)]$

These lemmas and this definition suffice to prove our main theorem.

Theorem 1 (MMS Security) $\forall T \forall C [\, \text{L}(T.C) \rightarrow \text{MMS\_secure}(T, C)\,]$.

References

[Bart77] Bartussek, A. W., and Parnas, D. L., "Using Traces to Write Abstract Specifications for Software Modules," UNC Report TR77-012, University of North Carolina, Chapel Hill, North Carolina, 1977.

[Hoff84] Hoffman, Daniel M., *Trace Specification of Communications Protocols*, UNC Report TR84-009, Department of Computer Science, University of North Carolina, Chapel Hill, North Carolina, 1984.

[Hoff86] Hoffman, Daniel M., and Snodgrass, Richard, "Trace Specification: Methodology and Models," UV Report DCS-53-IR, Department of Computer Science, University of Victoria, B.C., Canada, 1986.

[Land81] Landwehr, Carl E., "Formal Models for Computer Security," *ACM Computing Surveys* **13**: 247-278, September 1981.

[Land84] Landwehr, Carl E., Heitmeyer, Constance L., and McLean, John, "A Security Model for Military Message Systems," *ACM Transactions on Computer Systems*, **2**: 198-222, August 1984.

[McLean85] McLean, John, "A Formal Method for the Abstract Specification of Software," *Journal of the ACM*, **31**: 600-627, July 1984.

[McLean86] McLean, John, "Using Trace Specifications for Program Semantics and Verification," NRL Report 9033, Naval Research Laboratory, 1987.

[Meadows87] Meadows, Catherine, "A Method for Automatically Translating Trace Specifications into Prolog," Naval Research Laboratory, manuscript, 1987.

Appendix – The MMS model specification

Procedures

login

$nf(T) \rightarrow [L(T.\text{login}(u, d)) \leftrightarrow [\neg \text{logged\_in}(u, d, T) \wedge \text{user\_exists}(u, T) \wedge \text{ref\_secure}(T, \text{login}(u, d))]]$

$[nf(T) \wedge L(T.\text{login}(u, d))] \rightarrow [nf(T.\text{login}(u, d)) \leftrightarrow \neg \exists x\, \exists R\, T = R.\text{logout}(u, d)]$

$[nf(T) \wedge L(T.\text{login}(u, d)) \wedge T = R.\text{logout}(u, d))] \rightarrow T.\text{login}(u, d) \equiv T$

create_user

$nf(T) \rightarrow [L(T.\text{create\_user}(u, v, l)) \leftrightarrow [\text{RO}(u, T, \text{sso}) \wedge \text{user\_exists}(u, T) \wedge \text{logged\_in}(u, T) \wedge \neg \text{user\_exists}(v, T) \wedge \text{ref\_secure}(T, \text{create\_user}(u, v, l))]]$

$[nf(T) \wedge L(T.\text{create\_user}(u, v, l))] \rightarrow [nf(T.\text{create\_user}(u, v, l) \leftrightarrow \neg \exists u'\, \exists R\, [T = R.\text{delete\_user}(u', v) \wedge \text{CU}(v, R) = l]]$

$[nf(T) \wedge L(T.\text{create\_user}(u, v, l)) \wedge T = R.\text{delete\_user}(u, v) \wedge \text{CU}(v, T) = l] \rightarrow T.\text{create\_user}(u, v, l) \equiv R$

set_CU

$nf(T) \rightarrow [L(T.\text{set\_cu}(u, v, l)) \leftrightarrow [\text{RO}(u, T, \text{sso}) \wedge \text{user\_exists}(u, T) \wedge \text{logged\_in}(u, T) \wedge \text{user\_exists}(v, T) \wedge \neg \text{logged\_in}(v, T) \wedge \text{ref\_secure}(T, \text{set\_cu}(u, v, l))]]$

$[nf(T) \wedge L(T.\text{set\_cu}(u, v, l))] \rightarrow [nf(T.\text{set\_cu}(u, v, l)) \leftrightarrow [\text{CU}(v, T) \neq l \wedge \neg \exists S\, \exists u'\, \exists v'\, \exists l'\, T = S.\text{set\_cu}(u', v', l')]]$

$[nf(T) \wedge L(T.\text{set\_cu}(u, v, l)) \wedge \text{CU}(v, T) = l] \rightarrow T.\text{set\_cu}(u, v, l) \equiv T$

$[nf(T) \wedge L(T.\text{set\_cu}(u, v, l)) \wedge T = S.\text{set\_cu}(u', v', l')] \rightarrow T.\text{set\_cu}(u, v, l) \equiv S.\text{set\_cu}(u, v, l)]$

add_R

$nf(T) \rightarrow [L(T.\text{add\_R}(u, v, r)) \leftrightarrow [\text{RO}(u, T, \text{sso}) \wedge \text{logged\_in}(u, T) \wedge \text{user\_exists}(v, T) \wedge \text{ref\_secure}(T, \text{add\_R}(u, v, r))]]$

$[nf(T) \wedge L(T.\text{add\_R}(u, v, r))] \rightarrow [nf(T.\text{add\_R}(u, v, r)) \leftrightarrow [\neg \text{R}(v, T, r) \wedge \neg \exists u'\, \exists S\, T = S.\text{rm\_R}(u', v, r)]]$

$[nf(T) \wedge L(T.\text{add\_R}(u, v, r)) \wedge \text{R}(v, T, r)] \rightarrow T.\text{add\_R}(u, v, r) \equiv T$

$[nf(T) \wedge L(T.\text{add\_R}(u, v, r)) \wedge T = S.\text{rm\_R}(u', v, r)] \rightarrow T.\text{add\_R}(u, v, r)) \equiv S]$

rm_R

$nf(T) \rightarrow [L(T.\text{rm\_R}(u, v, r) \leftrightarrow [\text{logged\_in}(u, T) \wedge \text{RO}(u, T, \text{sso}) \wedge \text{user\_exists}(v, T) \wedge \neg \text{RO}(v, T, r) \wedge \text{ref\_secure}(T, \text{rm\_R}(u, v, r))]]$

$[nf(T) \wedge L(T.\text{rm\_R}(u, v, r)] \rightarrow [nf(T.\text{rm\_R}(u, v, r)) \leftrightarrow [\text{R}(v, T, r) \wedge \neg \exists u'\, \exists S\, T = S.\text{add\_R}(u', v, r)]]$

$[nf(T) \wedge L(T.\text{rm\_R}(u, v, r) \wedge \neg \text{R}(v, T, r)] \rightarrow T.\text{rm\_R}(u, v, r) \equiv T$

$[nf(T) \wedge L(T.\text{rm\_R}(u, v, r) \wedge T = S.\text{add\_R}(u', v, r)] \rightarrow T.\text{rm\_R}(u, v, r) \equiv S$

add_RO

$nf(T) \rightarrow [L(T.\text{add\_RO}(u, v, r)) \leftrightarrow [[\text{RO}(u, T, \text{sso}) \vee u = v] \wedge \text{logged\_in}(u, T) \wedge \text{R}(v, T, r) \wedge \text{user\_exists}(v, T) \wedge \text{ref\_secure}(T, \text{add\_RO}(u, v, r))]]$

$[nf(T) \wedge L(T.\text{add\_RO}(u, v, r))] \rightarrow [nf(T.\text{add\_RO}(u, v, r)) \leftrightarrow [\neg \text{RO}(v, T, r) \wedge \neg \exists u'\, \exists S\, T = S.\text{rm\_RO}(u', v, r)]]$

$[nf(T) \wedge L(T.\text{add\_RO}(u, v, r)) \wedge \text{RO}(v, T, r)] \rightarrow T.\text{add\_RO}(u, v, r) \equiv T$

$[nf(T) \wedge L(T.\text{add\_RO}(u, v, r)) \wedge T = S.\text{rm\_RO}(u', v, r)] \rightarrow T.\text{add\_RO}(u, v, r)) \equiv S]$

rm_RO

$nf(T) \rightarrow [L(T.\text{rm\_RO}(u, v, r) \leftrightarrow [\text{logged\_in}(u, T) \wedge [\text{RO}(u, T, \text{sso}) \vee u = v] \wedge \text{R}(v, T, r) \wedge \text{user\_exists}(v, T) \wedge \text{ref\_secure}(T, \text{rm\_RO}(u, v, r))]]$

$[nf(T) \wedge L(T.\text{rm\_RO}(u, v, r)] \rightarrow [nf(T.\text{rm\_RO}(u, v, r)) \leftrightarrow [\text{RO}(v, T, r) \wedge \neg \exists u'\, \exists S\, T = S.\text{add\_R}(u', v, r)]]$

$[nf(T) \wedge L(T.\text{rm\_RO}(u, v, r) \wedge \neg \text{RO}(v, T, r)] \rightarrow T.\text{rm\_RO}(u, v, r) \equiv T$

$[nf(T) \wedge L(T.\text{rm\_RO}(u, v, r) \wedge T = S.\text{add\_RO}(u', v, r)] \rightarrow T.\text{rm\_RO}(u, v, r) \equiv S$

delete_user

$nf(T) \rightarrow [L(T.\text{delete\_user}(u, v)) \leftrightarrow [\text{logged\_in}(u, T) \wedge \text{RO}(u, T, \text{sso}) \wedge \text{user\_exists}(v, T) \wedge \text{ref\_secure}(T, \text{delete\_user}(u, v))]]$

$[nf(T) \wedge L(T.\text{delete\_user}(u, v))] \rightarrow [nf(T.\text{delete\_user}(u, v) \leftrightarrow [\neg \exists u'\, \exists S\, T = S.\text{create\_user}(u', v)]$

$[nf(T) \wedge L(T.\text{delete\_user}(u, v)) \wedge T = S.\text{create\_user}(u', v)] \rightarrow T.\text{delete\_user}(u, v) \equiv S]$

add_AS

$[nf(T) \rightarrow [L(T.\text{add\_AS}(u, v, c, i, k)) \leftrightarrow [\text{logged\_in}(u, T) \wedge RO(u, T, sso) \wedge \text{ref\_exists}(i, T) \wedge [\text{role}(v) \vee \text{user\_exists}(v, T)] \wedge \text{ref\_secure}(T, \text{add\_AS}(u, v, c, i, k))]]$

$[nf(T) \wedge L(T.\text{add\_AS}(u, v, c, i, k))] \rightarrow [nf(T.\text{add\_AS}(u, v, c, i, k)) \leftrightarrow \forall S\, \text{after\_every\_ref}_{(3,3)}(S, \text{add\_AS}, \text{rm\_AS}, (4, i), 2, v, 3, c, 5, k; (4, i), 2, v, 3, c, 5, k, T) \wedge \neg\exists j\, \exists u'\, \exists S\, [T = S.\text{rm\_AS}(u', v, c, j, k) \wedge \text{coref}(i, T, j, S)]]$

$[nf(T) \wedge L(T.\text{add\_AS}(u, v, c, i, k)) \wedge T = S.\text{rm\_AS}(u', v, c, j, k) \wedge \text{coref}(i, T, j, S)] \rightarrow T.\text{add\_AS}(u, v, c, i, k) \equiv S]$

$[nf(T) \wedge L(T.\text{add\_AS}(u, v, c, i, k)) \wedge \neg\forall S\, \text{after\_every\_ref}_{(3,3)}(S, \text{add\_AS}, \text{rm\_AS}, (4, i), 2, v, 3, c, 5, k; (4, i), 2, v, 3, c, 5, k, T)] \rightarrow T.\text{add\_AS}(u, v, c, i, k) \equiv T]$

rm_AS

$[nf(T) \rightarrow [L(T.\text{rm\_AS}(u, v, c, i, k)) \leftrightarrow [\text{logged\_in}(u, T) \wedge RO(u, T, sso) \wedge \text{ref\_exists}(i, T) \wedge [\text{role}(v) \vee \text{user\_exists}(v, T)] \wedge \text{ref\_secure}(T, \text{rm\_AS}(u, v, c, i, k))]]$

$[nf(T) \wedge L(T.\text{rm\_AS}(u, v, c, i, k))] \rightarrow [nf(T.\text{rm\_AS}(u, v, c, i, k)) \leftrightarrow \exists S\, \text{some\_none\_ref}_{(3,3)}(S, \text{add\_AS}, \text{rm\_AS}, (4, i), 2, v, 3, c, 5, k; (4, i), 2, v, 3, c, 5, k, T) \wedge \neg\exists j\, \exists u'\, \exists S\, [T = S.\text{add\_AS}(u', v, c, j, k) \wedge \text{coref}(i, T, j, S)]]$

$[nf(T) \wedge L(T.\text{rm\_AS}(u, v, c, i, k)) \wedge T = S.\text{add\_AS}(u', v, c, j, k) \wedge \text{coref}(i, T, j, S)] \rightarrow T.\text{rm\_AS}(u, v, c, i, k) \equiv S]$

$[nf(T) \wedge L(T.\text{rm\_AS}(u, v, c, i, k)) \wedge \exists S\, \text{some\_none\_ref}_{(3,3)}(S, \text{add\_AS}, \text{rm\_AS}, (4, i), 2, v, 3, c, 5, k; (4, i), 2, v, 3, c, 5, k, T)] \rightarrow T.\text{rm\_AS}(u, v, c, i, k) \equiv T]$

current_R

$nf(T) \rightarrow [L(T.\text{current\_R}(u, v, r)) \leftrightarrow [\text{logged\_in}(u, T) \wedge [RO(u, T, sso) \vee u = v] \wedge \text{user\_exists}(v, T) \wedge \text{ref\_secure}(T, \text{current\_R}(u, v, r))]]$

$\neg nf(T.\text{current\_R}(u, v))$

$[nf(T) \wedge L(T.\text{current\_R}(u, v, r))] \rightarrow T.\text{current\_R}(u, v, r) \equiv T$

$[nf(T) \wedge L(T.\text{current\_R}(u, v, r))] \rightarrow [[V(T.\text{current\_R}(u, v, r)) = \text{True} \leftrightarrow R(v, T, r)] \wedge [V(T.\text{current\_R}(u, v, r)) = \text{False} \leftrightarrow \neg R(v, T, r)]]$

current_RO

$nf(T) \rightarrow [L(T.\text{current\_RO}(u, v, r)) \leftrightarrow [\text{logged\_in}(u, T) \wedge [RO(u, T, sso) \vee u = v] \wedge \text{user\_exists}(v, T) \wedge \text{ref\_secure}(T, \text{current\_RO}(u, v, r))]]$

$\neg nf(T.\text{current\_RO}(u, v))$

$[nf(T) \wedge L(T.\text{current\_RO}(u, v, r))] \rightarrow T.\text{current\_RO}(u, v, r) \equiv T$

$[nf(T) \wedge L(T.\text{current\_RO}(u, v, r))] \rightarrow [[V(T.\text{current\_RO}(u, v, r)) = \text{True} \leftrightarrow RO(v, T, r)] \wedge [V(T.\text{current\_RO}(u, v, r)) = \text{False} \leftrightarrow \neg RO(v, T, r)]]$

current_CU

$nf(T) \rightarrow [L(T.\text{current\_CU}(u, v, r)) \leftrightarrow [\text{logged\_in}(u, T) \wedge [RO(u, T, sso) \vee u = v] \wedge \text{user\_exists}(v, T) \wedge \neg$ [illegible] $, \text{current\_CU}(u, v, r))]]$

$\neg nf(T.\text{current\_CU}(u, v))$

$[nf(T) \wedge L(T.\text{current\_CU}(u, v, r))] \rightarrow T.\text{current\_CU}(u, v, r) \equiv T$

$[nf(T) \wedge L(T.\text{current\_CU}(u, v, l))] \rightarrow [V(T.\text{current\_CU}(u, v)) = CU(v, T)]$

set_CCR

$nf(T) \rightarrow [L(T.\text{set\_CCR}(u, r, x)) \leftrightarrow [\text{logged\_in}(u, T) \wedge \text{user\_exists}(u, T) \wedge \text{container}(r, T) \wedge \text{ref\_exists}(r, T) \wedge AS(u, \text{set\_CCR}, r, 2, T) \wedge \text{ref\_secure}(T, \text{set\_CCR}(u, r, x))]]$

$[nf(T) \wedge L(T.\text{set\_CCR}(u, r, x))] \rightarrow [nf(T.\text{set\_CCR}(u, r, x)) \leftrightarrow [CCR(r, T) \neq x \wedge \neg\exists S\, \exists v\, \exists y\, T = S.\text{set\_CCR}(v, r, y)]]$

$[nf(T) \wedge L(T.\text{set\_CCR}(u, r, x)) \wedge CCR(r, T) = x] \rightarrow T.\text{set\_CCR}(u, r, x) \equiv T$

$[nf(T) \wedge L(T.\text{set\_CCR}(u, r, x)) \wedge T = S.\text{set\_CCR}(v, r, y)] \rightarrow T.\text{set\_CCR}(u, r, x) \equiv S.\text{set\_CCR}(u, r, x)]$

downgrade

$nf(T) \rightarrow [L(T.\text{downgrade}(u, r, l) \rightarrow [RO(u, \text{downgrader}, T) \wedge \text{logged\_in}(u, T) \wedge \text{user\_exists}(u, T) \wedge \text{ref\_exists}($ [illegible] $\wedge CE(r, T) > l \wedge \forall t\, \forall k\, [H(t, T, r, k) \rightarrow CE(t, T) \leq l] \wedge \text{callname}(C) = \text{downgrade} \wedge AS(u, \text{downgrade}, r, 2, T) \wedge \text{ref\_secure}(T, \text{downgrade}(u, r, l))]]$

$[nf(T) \wedge L(T.\text{downgrade}(u, r, l)] \rightarrow nf(T.\text{downgrade}(u, r, l))$

release

$nf(T) \rightarrow [L(T.\text{release}(u, r) \leftrightarrow [RO(u, \text{releaser}, T) \wedge \text{user\_exists}(u, T) \wedge \text{logged\_in}(u, T) \wedge \text{ref\_exists}(r, T) \wedge T(r, T) = DM \wedge AS(u, \text{release}, r, 2, T) \wedge \text{ref\_secure}(T, \text{release}(u, r))]]$

$[nf(T) \wedge L(T.\text{release}(u, r))] \rightarrow nf(T.\text{release}(u, r))$

logout

$nf(T) \rightarrow [L(T.\text{logout}(u, d)) \leftrightarrow [\text{logged\_in}(u, d, T) \wedge \neg\exists r\, \exists k\, H(d, T, r, k) \wedge \text{ref\_secure}(T, \text{logout}(u, d))]]$

$[nf(T) \wedge L(T.\text{logout}(u, d))] \rightarrow [nf(T.\text{logout}(u, d)) \leftrightarrow \neg\exists S\, T = S.\text{login}(u, d)]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{logout}(u, d)) \wedge T = S.\mathrm{login}(u, d)] \rightarrow T.\mathrm{logout}(u, d) \doteq S]$

current_AS

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathrm{current\_AS}(u, v, c, r, k)) \leftrightarrow [\mathrm{logged\_in}(u, T) \wedge \mathrm{RO}(u, T, \mathrm{sso}) \wedge \mathrm{user\_exists}(v, T) \wedge \mathrm{ref\_exists}(r, T) \wedge \mathrm{ref\_secure}(T, \mathrm{current\_AS}(u, v, c, r, k))]]$

$\neg\mathrm{nf}(T.\mathrm{current\_AS}(u, v, c, r, k))$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{current\_AS}(u, v, c, r, k))] \rightarrow T.\mathrm{current\_AS}(u, v, c, r, k) \equiv T$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{current\_AS}(u, v, c, r, k))] \rightarrow [[\mathrm{V}(T.\mathrm{current\_AS}(u, v, c, r, k)) = \mathrm{True} \leftrightarrow \mathrm{AS}(v, c, r, k, T)] \wedge [\mathrm{V}(T.\mathrm{current\_AS}(u, v, c, r, k)) = \mathrm{False} \leftrightarrow \neg\mathrm{AS}(v, c, r, k, T)]]$

current_CE

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathrm{current\_CE}(u, r)) \leftrightarrow [\mathrm{logged\_in}(u, T) \wedge \mathrm{RO}(u, T, \mathrm{sso}) \wedge \mathrm{user\_exists}(v, T) \wedge \mathrm{ref\_exists}(r, T) \wedge \mathrm{ref\_secure}(T, \mathrm{current\_CE}(u, r))]]$

$\neg\mathrm{nf}(T.\mathrm{current\_CE}(u, r))$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{current\_CE}(u, r))] \rightarrow T.\mathrm{current\_CE}(u, r) \equiv T$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{current\_CE}(u, r))] \rightarrow [\mathrm{V}(T.\mathrm{current\_CE}(u, r)) = \mathrm{CE}(r, T)$

current_CCR

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathrm{current\_CCR}(u, r)) \leftrightarrow [\mathrm{logged\_in}(u, T) \wedge \mathrm{RO}(u, T, \mathrm{sso}) \wedge \mathrm{ref\_exists}(r, T) \wedge \mathrm{ref\_secure}(T, \mathrm{current\_CCR}(u, r))]]$

$\neg\mathrm{nf}(T.\mathrm{current\_CCR}(u, r))$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{current\_CCR}(u, r))] \rightarrow T.\mathrm{current\_CCR}(u, r) \equiv T$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{current\_CCR}(u, r))] \rightarrow [\mathrm{V}(T.\mathrm{current\_CCR}(u, r)) = \mathrm{CCR}(r, T)$

current_TY

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathrm{current\_TY}(u, r)) \leftrightarrow [\mathrm{logged\_in}(u, T) \wedge \mathrm{RO}(u, T, \mathrm{sso}) \wedge \mathrm{ref\_exists}(r, T) \wedge \mathrm{ref\_secure}(T, \mathrm{current\_TY}(u, r))]]$

$\neg\mathrm{nf}(T.\mathrm{current\_TY}(u, r))$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{current\_TY}(u, r))] \rightarrow T.\mathrm{current\_TY}(u, r) \equiv T$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{current\_TY}(u, r))] \rightarrow [\mathrm{V}(T.\mathrm{current\_TY}(u, r)) = \mathrm{T}(r, T)]$

set_CE

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathrm{set\_CE}(u, d, l)) \leftrightarrow [\mathrm{logged\_in}(u, T) \wedge \mathrm{AS}(u, \mathrm{set\_CE}, d, 2, T) \wedge \mathrm{device}(d) \wedge l \leq \mathrm{CD}(d, T) \wedge \mathrm{ref\_secure}(T, \mathrm{set\_CE}(u, d, l))]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{set\_CE}(u, d, l))] \rightarrow [\mathrm{nf}(T.\mathrm{set\_CE}(u, d, l)) \leftrightarrow [\neg\exists u' \exists l' \exists S\, T = S.\mathrm{set\_CE}(u', d, l') \wedge \forall C \forall R [[\mathrm{prefix}(R.C, T) \wedge \mathrm{callname}(C) = \mathrm{set\_CE} \wedge \mathrm{coref}(\mathrm{arg}(C, 2), R, d, T)] \rightarrow [\mathrm{arg}(C, 2) \neq l \vee \exists C' \exists S [\mathrm{callname}(C') = \mathrm{set\_CE} \wedge \mathrm{coref}(\mathrm{arg}(C', 2), S, d, T) \wedge \mathrm{prefix}(R.C, S.C') \wedge \mathrm{arg}(C', 2) \neq l]]]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{set\_CE}(u, d, l)) \wedge T = S.\mathrm{set\_CE}(u', d, l')] \rightarrow T.\mathrm{set\_CE}(u, d, l) \equiv S.\mathrm{set\_CE}(u, d, l)$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{set\_CE}(u, d, l)) \wedge \exists C \exists R [\mathrm{prefix}(R.C, T) \wedge \mathrm{callname}(C) = \mathrm{set\_CE} \wedge \mathrm{coref}(\mathrm{arg}(C, 2), R, d, T) \wedge \mathrm{arg}(C, 2) = l \wedge \neg\exists C' \exists S [\mathrm{callname}(C') = \mathrm{set\_CE} \wedge \mathrm{coref}(\mathrm{arg}(C', 2), S, d, T) \wedge \mathrm{prefix}(R.C, S.C') \wedge \mathrm{arg}(C', 2) \neq l]]] \rightarrow T.\mathrm{set\_CE}(u, d, l) \equiv T$

set_CD

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathrm{set\_CD}(u, d, l)) \leftrightarrow [\mathrm{logged\_in}(u, T) \wedge \mathrm{RO}(u, T, \mathrm{sso}) \wedge \mathrm{CE}(d, T) \leq l \wedge \mathrm{device}(d) \wedge \mathrm{ref\_secure}(T, \mathrm{set\_CD}(u, d, l))]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{set\_CD}(u, d, l))] \rightarrow [\mathrm{nf}(T.\mathrm{set\_CD}(u, d, l)) \leftrightarrow [\neg\exists u' \exists l' \exists S\, T = S.\mathrm{set\_CD}(u', d, l') \wedge \forall C \forall R [[\mathrm{prefix}(R.C, T) \wedge \mathrm{callname}(C) = \mathrm{set\_CD} \wedge \mathrm{coref}(\mathrm{arg}(C, 2), R, d, T)] \rightarrow [\mathrm{arg}(C, 2) \neq l \vee \exists C' \exists S [\mathrm{callname}(C') = \mathrm{set\_CD} \wedge \mathrm{coref}(\mathrm{arg}(C', 2), S, d, T) \wedge \mathrm{prefix}(R.C, S.C') \wedge \mathrm{arg}(C', 2) \neq l]]]]$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{set\_CD}(u, d, l)) \wedge T = S.\mathrm{set\_CD}(u', d, l')] \rightarrow T.\mathrm{set\_CD}(u, d, l) \equiv S.\mathrm{set\_CD}(u, d, l)$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{set\_CD}(u, d, l)) \wedge \exists C \exists R [\mathrm{prefix}(R.C, T) \wedge \mathrm{callname}(C) = \mathrm{set\_CD} \wedge \mathrm{coref}(\mathrm{arg}(C, 2), R, d, T) \wedge \mathrm{arg}(C, 2) = l \wedge \neg\exists C' \exists S [\mathrm{callname}(C') = \mathrm{set\_CD} \wedge \mathrm{coref}(\mathrm{arg}(C', 2), S, d, T) \wedge \mathrm{prefix}(R.C, S.C') \wedge \mathrm{arg}(C', 2) \neq l]]] \rightarrow T.\mathrm{set\_CD}(u, d, l) \equiv T$

current_CD

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathrm{current\_CD}(u, d)) \leftrightarrow [\mathrm{logged\_in}(u, T) \wedge \mathrm{RO}(u, T, \mathrm{sso}) \wedge \mathrm{device}(d)]]$

$\neg\mathrm{nf}(T.\mathrm{current\_CD}(u, d))$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{current\_CD}(u, d))] \rightarrow T.\mathrm{current\_CD}(u, d) \equiv T$

$[\mathrm{nf}(T) \wedge \mathrm{L}(T.\mathrm{current\_CD}(u, d))] \rightarrow \mathrm{V}(T.\mathrm{current\_CD}(u, d)) = \mathrm{CD}(d, T)]$

display

$\mathrm{nf}(T) \rightarrow [\mathrm{L}(T.\mathrm{display}(u, r, f, d)) \leftrightarrow [\mathrm{logged\_in}(u, T) \wedge \mathrm{ref\_exists}(r, T) \wedge \mathrm{device}(d) \wedge \mathrm{CU}(u, T) \geq \mathrm{CE}(r, T) \wedge \mathrm{CD}(d, T) \geq \mathrm{CE}(r, T) \wedge \mathrm{AS}(u, \mathbf{display}, r, 2, T) \wedge [f = \mathrm{value} \vee f = \mathrm{direct\_ref} \vee f = \mathrm{classification} \vee f = \mathrm{type} \vee f = \mathrm{ccr\_value}] \wedge [f = \mathrm{value} \rightarrow [\mathrm{part\_of}(r, d, T) \wedge \exists S\, T = S.\mathbf{display}(u, r, \mathrm{classification}, d)]] \wedge \forall z [[f = \mathrm{direct\_ref} \wedge \mathrm{based\_on}(r, z, T) \wedge \mathrm{CCR}(z, T) = \mathrm{True}] \rightarrow$

$CU(u, T) \geq CE(x, T)]] \wedge ref\_secure(T, display(u, r, f, d))]]$

$[nf(T) \wedge L(T.display(u, r, f, d))] \rightarrow [nf(T.display(u, r, f, d)) \leftrightarrow \forall S \forall C [[prefix(S.C, T) \wedge coref(arg(C, 2), S, r, T) \wedge callname(C) = display \wedge arg(C, 3) = f \wedge arg(C, 4) = d] \rightarrow \exists R \exists D [prefix(S, R) \wedge [[callname(D) = display \wedge arg(D, 4) = d \wedge coref(arg(D, 2), R, r, T) \wedge arg(D, 3) \neq f] \vee [callname(D) = delete\_ref \wedge coref(arg(D, 2), R, r, T)] \vee [arg(D, 4) = d \wedge callname(D) = identify]]]]$

$[nf(T) \wedge L(T.display(u, r, f, d)) \wedge \exists S \exists C [prefix(S.C, T) \wedge coref(arg(C, 2), S, r, T) \wedge callname(C) = display \wedge arg(C, 3) = f \wedge arg(C, 4) = d \wedge \neg \exists R \exists D [prefix(S, R) \wedge [[callname(D) = display \wedge arg(D, 4) = d \wedge coref(arg(D, 2), R, r, T) \wedge arg(D, 3) \neq f] \vee [callname(D) = delete\_ref \wedge coref(arg(D, 2), R, r, T)] \vee [arg(D, 4) = d \wedge callname(D) = identify]]]] \rightarrow T.display(u, r, f, d) \equiv T$

obj_create

$nf(T) \rightarrow [L(T.obj\_create(u, k, l, p)) \leftrightarrow [logged\_in(u, T) \wedge string(p) \wedge \neg \exists S [prefix(S, T) \wedge ref\_exists(k, S)] \wedge ref\_secure(T, obj\_create(u, k, l, p))]]$

$[nf(T) \wedge L(T.obj\_create(u, k, l, p))] \rightarrow nf(T.obj\_create(u, k, l, p))$

cont_create

$nf(T) \rightarrow [L(T.cont\_create(u, k, l)) \leftrightarrow [logged\_in(u, T) \wedge \neg \exists S [prefix(S, T) \wedge ref\_exists(k, T)] \wedge ref\_secure(T, cont\_create(u, k, l))]]$

$[nf(T) \wedge L(T.cont\_create(u, k, l))] \rightarrow nf(T.cont\_create(u, k, l))$

insert

$nf(T) \rightarrow [L(T.insert(u, x, y, k)) \leftrightarrow [l_{ogged\_in}(u, T) \wedge [logged\_in(u, y, T) \rightarrow CU(u, T) \geq CE(y, T)] \wedge ref\_exists(x, T) \wedge ref\_exists(y, T) \wedge CE(x, T) \leq CE(y, T) \wedge \neg part\_of(y, x, T) \wedge k \geq 0 \wedge container(y) \wedge AS(u, insert, x, 2, T) \wedge AS(u, insert, y, 3, T) \wedge ref\_secure(T, insert(u, x, y, k)) \wedge [k = 1 \vee \exists s H(s, T, y, k - 1)]]]$

$[nf(T) \wedge L(T.insert(u, x, y, k))] \rightarrow nf(T.insert(u, x, y, k))$

remove

$nf(T) \rightarrow [L(T.remove(u, x, y, k)) \leftrightarrow [logged\_in(u, T) \wedge ref\_exists(x, T) \wedge k \geq 0 \wedge AS(u, remove, x, 2, T) \wedge AS(u, remove, y, 3, T) \wedge coref(y, T, x : k, T) \wedge ref\_secure(T, remove(u, x, y, k))]]$

$[nf(T) \wedge L(T.remove(u, x, y, k))] \rightarrow nf(T.remove(u, x, y, k))$

delete_ref

$nf(T) \rightarrow [L(T.delete\_ref(u, r)) \leftrightarrow [logged\_in(u, T) \wedge ref\_exists(r, T) \wedge AS(u, delete\_ref, r, 2, T) \wedge ref\_secure(T, delete\_ref(u, r))]]$

$[nf(T) \wedge L(T.delete\_ref(u, r))] \rightarrow nf(T.delete\_ref(u, r))$

identify

$nf(T) \rightarrow [L(T.identify(u, N, f, d)) \leftrightarrow [logged\_in(u, T) \wedge device(d) \wedge user(N) \wedge [f = user\_id \vee f = clearance \vee f = access\_set \vee f = role\_set \vee f = roles] \wedge ref\_secure(T, identify(u, N, f, d))]]$

$[nf(T) \wedge L(T.identify(u, N, f, d))] \rightarrow [nf(T.identify(u, N, f, d)) \leftrightarrow \forall S \forall C [[prefix(S.C, T) \wedge callname(C) = identify \wedge arg(C, 2) = N \wedge arg(C, 3) = f \wedge arg(C, 4) = d] \rightarrow \exists R \exists D [prefix(S, R) \wedge arg(D, 4) = d \wedge [[callname(D) = identify \wedge arg(D, 2) = N \wedge arg(D, 3) \neq f] \vee callname(D) = display]]$

$[nf(T) \wedge L(T.identify(u, N, f, d)) \wedge \exists S \exists C [prefix(S.C, T) \wedge callname(C) = identify \wedge arg(C, 2) = N \wedge arg(C, 3) \neq f \wedge arg(C, 4) = d \wedge \neg \exists R \exists D [prefix(S.C, R) \wedge arg(D, 4) = d \wedge [[arg(D, 2) = N \wedge callname(D) = identify \wedge arg(D, 3) = f] \vee callname(D) = display]]] \rightarrow T.identify(u, N, f, d) \equiv T$

Definitions of predicates

Prefixes

$prefix(e, T)$

$[prefix(S, T) \wedge T = S.C.R] \rightarrow prefix(S.C, T)$

$prefix(S, T) \rightarrow [S = e \vee \exists R \exists C [S = R.C \wedge prefix(R, T)]]$

References

$direct\_ref(k) \wedge \forall r \exists k [direct\_ref(r) \rightarrow r = k]$

$direct\_ref(r) \rightarrow reference(r)$

$reference(r) \rightarrow reference(r : k)$

$reference(r) \rightarrow [direct\_ref(r) \vee \exists s \exists k [reference(s) \wedge r = s : k]]$

Coreference

$coref(r, T, r, T)$

$coref(r, T, s, R) \rightarrow coref(s, R, r, T)$

$\text{coref}(r, T, s, R) \rightarrow [\text{coref}(s, R, t, S) \rightarrow \text{coref}(r, T, t, S)]$

$\text{callname}(C) = \text{insert} \rightarrow [\text{coref}(\text{arg}(C, 2), T, \text{arg}(C, 3) : \text{arg}(C, 4), T.C \wedge [0 < k < \text{arg}(C, 4) \rightarrow \text{coref}(\text{arg}(C, 3) : k, T, \text{arg}(C, 3) : k, T.C)] \wedge [[k > \text{arg}(C, 4) \wedge \text{ref\_exists}(\text{arg}(C, 3) : k, T)] \rightarrow \text{coref}(\text{arg}(C, 3) : k, T, \text{arg}(C, 3) : (k + 1), T.C)]$

$[[\text{callname}(C) = \text{delete\_ref} \wedge \text{coref}(\text{arg}(C, 2), S, s : k, T)] \vee [\text{callname}(C) = \text{remove} \wedge \text{coref}(\text{arg}(C, 3), S, \text{arg}(C, 2) : \text{arg}(C, 4), T) \wedge k = \text{arg}(C, 4) \wedge s = \text{arg}(C, 3)] \rightarrow [0 < n < k \rightarrow \text{coref}(s : n, T, s : n, T.C)] \wedge [[n > k \wedge \text{ref\_exists}(s : n, T)] \rightarrow \text{coref}(s : n, T, s : (n - 1), T.C)]]$

$[\text{callname}(C) \neq \text{delete\_ref} \wedge \text{callname}(C) \neq \text{insert}] \rightarrow [\text{coref}(r, T, t, S) \rightarrow \text{coref}(r, T, t, S.C)]$

$\text{coref}(r, T, s : k, T) \rightarrow \exists C \exists S [\text{prefix}(S.C, T) \wedge \text{callname}(C) = \text{insert} \wedge \text{coref}(\text{arg}(C, 2), S, r, T) \wedge \text{coref}(\text{arg}(C, 3), S, s, T)]$

part_of

$\text{coref}(r, T, s, T) \rightarrow \text{part\_of}(r, s, T)$

$[\text{part\_of}(r, s, T) \wedge \text{ref\_exists}(r : k, T)] \rightarrow \text{part\_of}(r : k, s, T)$

Entities

$\forall r \forall s \forall T \forall R [\text{coref}(r, T, s, R) \rightarrow E(r, T) = E(s, R)]$

$\forall r \forall T [\text{ref\_exists}(r, T) \rightarrow \exists y\, y = E(r, T)$

$\text{entity\_exists}(x, T) \leftrightarrow \exists r\, x = E(r, T)$

$\text{entity}(x) \leftrightarrow \exists T \text{entity\_exists}(x, T)$

$[\text{entity}(x) \wedge \text{entity}(y) \rightarrow [x = y \leftrightarrow \exists S \exists T \exists r_1 \exists r_2 [E(r_1, S) = x \wedge E(r_2, T) = y \wedge \text{coref}(r_1, S, r_2, T)]]$

$\forall x \forall T [\text{entity\_exists}(x, T) \rightarrow \exists z [z = \text{value}(x, T) \wedge \text{string}(z)]]$

Basing References on Entities

$\forall r \forall k \forall e [\text{ref\_exists}(r : k, T) \wedge E(r, T) = e] \rightarrow \text{based\_on}(r : k, e, T)$

$\forall r \forall k \forall e [\text{ref\_exists}(r : k, T) \wedge \text{based\_on}(r, e, T)] \rightarrow \text{based\_on}(r : k, e, T)$

Potential Modification

$\text{e\_equiv}(S, T, x) \leftrightarrow \exists r_1 \exists r_2 [\text{entity}(x) \wedge E(r_1, S) = x \wedge E(r_2, T) = x \wedge \forall u \forall c \forall k [AS(u, c, r_1, k, S) \leftrightarrow AS(u, c, r_2, k, T)] \wedge [CCR(r_1, S) = \text{True} \leftrightarrow CCR(r_2, T)] \wedge [TY(x, S) = TY(x, T)] \wedge \text{value}(x, S) = \text{value}(x, T) \wedge \forall s_1 \forall s_2 [\text{coref}(s_1, S, s_2, T) \rightarrow [\text{part\_of}(s_1, r_1, S) \leftrightarrow \text{part\_of}(s_2, r_2, T)]] \wedge CE(r_1, S) = CE(r_2, T) \wedge [[\text{device}(r_1) \wedge \text{device}(r_2)] \rightarrow CD(r_1, S) = CD(r_2, T)]]$

$\text{o\_equiv}(S, T, y) \leftrightarrow [S \equiv T \wedge \forall x [x \neq y \leftrightarrow \text{value}(x, S) = \text{value}(x, T)]$

$\text{p\_modify}(C, r, T, y) \leftrightarrow \exists S [L(T.C) \wedge \text{ref\_exists}(r, T) \wedge \text{entity\_exists}(y, T) \wedge S \equiv T \wedge \neg \text{e\_equiv}(S, S.C, E(r, S)) \wedge [y = E(r, T) \vee \exists S_1 \exists S_2 [S \equiv S_1 \wedge \text{o\_equiv}(S_1, S_2, y) \wedge \neg \text{e\_equiv}(S_2, S_2.C, E(r, S_2))]]]$

Security properties

$\text{state\_secure}(T) \leftrightarrow [\forall r \forall s \forall k [H(r, T, s, k) \rightarrow CE(s, T) \leq CE(r, T)] \wedge \forall o \forall u \forall k \forall r [[H(o, T, r, k) \wedge \text{logged\_in}(u, o, T)] \rightarrow CU(u, T) \geq CE(r, T)] \wedge \forall o \forall x [D(o, x, \text{value}, T) \rightarrow D(o, x, \text{classification}, T)] \wedge \forall u [RO(u, T, r) \rightarrow R(u, T, r)] \wedge \forall o [CD(o, T) \geq CE(o, T)]]$

$\text{acc\_secure}(T, C) \rightarrow \forall k [\text{reference}(\text{arg}(C, k)) \rightarrow [AS(\text{arg}(C, 1), \text{callname}(C), \text{arg}(C, k), k, T) \vee \exists r [RO(\text{arg}(C, 1), T, r) \wedge AS(r, \text{callname}(C), \text{arg}(C, k), k, T)]]]$

$\text{copy\_secure}(T, C) \leftrightarrow \forall r \forall y [\text{p\_modify}(C, r, T, y) \rightarrow \exists x [y = E(x, T) \wedge CE(x, T) \leq CE(r, T)]$

$\text{CCR\_secure}(T, C) \leftrightarrow \forall u \forall r \forall z \forall y [[\exists k\, r = \text{arg}(C, k) \wedge \text{based\_on}(r, y, T) \wedge \exists s [y = E(s, T) \wedge CCR(s, T) \wedge \text{p\_modify}(C, z, T, E(r, T))] \rightarrow CU(u, T) \geq CE(z, T)]$

$\text{trans\_secure}(T, C) \leftrightarrow \forall o \forall x \forall z \forall u [[\text{logged\_in}(u, o, T.C) \wedge \text{direct\_ref}(x) \wedge D(o, x, \text{direct\_ref}, T.C)] \rightarrow \forall s [\exists r \exists k [\text{arg}(C, k) = r \wedge E(r, T) = E(x, T) \wedge \text{based\_on}(r, z, T) \wedge E(s, T) = z \wedge CCR(s, T)] \rightarrow CU(u, T) \geq CE(s, T)]]]$

$\text{ref\_secure}(T, C) \leftrightarrow \forall u \forall r \forall k [\text{reference}(\text{arg}(C, k)) \rightarrow [CU(\text{arg}(C, 1), T) \geq CE(\text{arg}(C, K), T) \wedge [\exists s [\text{based\_on}(\text{arg}(C, k), E(s, T)) \wedge CCR(s, T)] \rightarrow CU(\text{arg}(C, 1), T) \geq CE(s, T)]]]$

$\text{set\_secure}(T, C) \leftrightarrow [\forall o \forall x [[CD(o, T) \neq CD(o, T.C) \vee CU(x, T) \neq CU(x, T.C) \vee \exists l \neg [R(x, T, l) \equiv R(x, T.C, l)]] \rightarrow RO(\text{arg}(C, 1), T, \text{sso})] \wedge [\forall u [\exists l \neg [RO(u, T, l) \equiv RO(u, T.C, l)] \rightarrow [u = \text{arg}(C, 1) \vee RO(u, T, \text{sso})]]]$

$\text{downgrade\_secure}(T, C) \leftrightarrow [[\neg \text{device}(x) \wedge CE(x, T) > CE(x, T.C)] \rightarrow [\text{callname}(C) = \text{downgrade} \wedge RO(\text{arg}(C, 1), T, \text{downgrader})]]$

$\text{release\_secure}(T, C) \leftrightarrow [[T(r, T) = RM \rightarrow [T(r, T.C) = RM \wedge \forall u [RE(r, T, u) \rightarrow RE(r, T.C, u)]] \wedge [[T(r, T) \neq RM \wedge T(r, T.C) = RM] \rightarrow [RE(r, T.C, \text{arg}(C, 1)) \wedge \text{callname}(C) = \text{release} \wedge \text{arg}(C, 2) = r \wedge RO(\text{arg}(C, 1), T, \text{releaser}) \wedge T(r, T) = DM]]]$

$\text{MMS\_secure}(T, C) \leftrightarrow [\text{acc\_secure}(T, C) \wedge \text{copy\_secure}(T, C) \wedge \text{CCR\_secure}(T, C) \wedge \text{trans\_secure}(T, C) \wedge \text{set\_secure}(T, C) \wedge \text{downgrade\_secure}(T, C) \wedge \text{release\_secure}(T, C) \wedge \text{state\_secure}(T)]$

Other predicates

$$\text{some\_none}_{(k_1, k_2)}(S, c_1, c_2, m_1, x_1, \ldots, m_{k_1}, x_{k_1}; n_1, y_1, \ldots, n_{k_2}, y_{k_2}, T) \leftrightarrow \exists C_1 [\text{prefix}(S.C_1, T) \wedge \text{callname}(C_1) = c_1 \wedge \bigwedge_{i=1}^{k_1} \{\text{arg}(C_1, m_i) = x_i\} \wedge \neg \exists R \exists C_2 [\text{prefix}(S.C_1, R) \wedge \text{prefix}(R.C_2, T) \wedge \text{callname}(C_2) = c_2 \wedge \bigwedge_{i=1}^{k_2} \{\text{arg}(C_2, n_i) = y_i\}]]$$

$$\text{after\_every}_{(k_1,k_2)}(S, c_1, c_2, m_1, x_1, \ldots, m_{k_1}, x_{k_1}; n_1, y_1, \ldots, n_{k_2}, y_{k_2}, T) \leftrightarrow \neg\text{some\_none}_{(k_1,k_2)}(S, c_1, c_2, m_1, x_1, \ldots, m_{k_1}, x_{k_1}; n_1, y_1, \ldots, n_{k_2}, y_{k_2}, T)$$

$$\text{in\_after}_k(S, c, m_1, x_1, \ldots, m_k, x_k, T) \leftrightarrow \exists C\, \exists R\, [S = R.C \wedge \text{prefix}(S, T) \wedge \text{callname}(C) = c \wedge \bigwedge_{i=1}^{k} \{arg(C, i) = x_i\}]$$

$$\text{user\_exists}(u, T) \leftrightarrow [u = \text{Root} \vee \exists S\, \text{some\_none}_{(1,1)}(S, \text{create\_user}, \text{delete\_user}, 2, u; 2, u, T)]$$

$$\text{logged\_in}(u, T) \leftrightarrow \exists d\, \text{logged\_in}(u, d, T)$$

$$\text{logged\_in}(u, d, T) \leftrightarrow \exists R\, [\text{user\_exists}(u, T) \wedge \text{some\_none}_{(2,2)}(R, \text{login}, \text{logout}, 1, u, 2, d; 1, u, , 2, d, T)]$$

$$\text{some\_none\_ref}_{(k_1,k_2)}(S, (m, r), c_1, c_2, m_1, x_1, \ldots, m_{k_1}, x_{k_1}; n_1, y_1, \ldots, (n, r), n_{k_2}, y_{k_2}, T) \leftrightarrow \exists s_1\, \exists C_1 [\text{prefix}(S.C_1, T) \wedge \text{arg}(C_1, m) = s_1 \wedge \text{coref}(s_1, S, r, T) \wedge \text{callname}(C_1) = c_1 \wedge \bigwedge_{i=1}^{k_1} \{\text{arg}(C_1, m_i) = x_i\} \wedge \neg\exists s_2\, \exists R\, \exists C_2 [\text{prefix}(S.C_1, R) \wedge \text{prefix}(R.C_2, T) \wedge \text{arg}(C_2, n) = s_2 \wedge \text{coref}(s_2, R, r, T) \wedge \text{callname}(C_2) = c_2 \wedge \bigwedge_{i=1}^{k_2} \{\text{arg}(C_2, n_i) = y_i\}]]$$

$$\text{after\_every\_ref}_{(k_1,k_2)}(S, (m, r), c_1, c_2, m_1, x_1, \ldots, m_{k_1}, x_{k_1}; (n, r), n_1, y_1, \ldots, n_{k_2}, y_{k_2}, T) \leftrightarrow \neg\text{some\_none\_ref}_{(k_1,k_2)}(S, (m, r), c_1, c_2, m_1, x_1, \ldots, m_{k_1}, x_{k_1}; (n, r), n_1, y_1, \ldots, n_{k_2}, y_{k_2}, T)$$

$$\text{in\_after\_ref}_k(S, c, (m, r), m_1, x_1, \ldots, m_k, x_k, T) \leftrightarrow \exists s\, \exists C\, \exists R\, [S = R.C \wedge \text{prefix}(S, T) \wedge \text{callname}(C) = c \wedge \text{coref}(s, R, r, T) \wedge \text{arg}(C, m) = s \wedge \bigwedge_{i=1}^{k} \{arg(C, i) = x_i\}]$$

$$\text{container}(r) \leftrightarrow \exists k\, \exists S\, \exists C\, [\text{coref}(k, S, r, T) \wedge \text{arg}(C, 2) = k \wedge \text{prefix}(S.C, T) \wedge \text{callname}(C) = \text{cont\_create} \wedge \text{ref\_exists}(r, T) \wedge \forall R\, \forall C_0\, [[\text{callname}(C_0) = \text{cont\_create} \vee \text{callname}(C_0) = \text{obj\_create}] \wedge \text{arg}(C_0, k) = k \wedge \text{prefix}(R.C_0, T)] \rightarrow \text{prefix}(R.C_0, S.C)]]$$

$$\text{ref\_exists}(r, T) \leftrightarrow \exists S\, \exists C\, [\text{some\_none\_ref}_{(0,0)}(S, c, \text{delete\_ref}, (2, r); (2, r), T) \wedge [c = \text{obj\_create} \vee c = \text{cont\_create}]]$$

$$\text{RO}(v, T, r) \leftrightarrow [[v = \text{Root} \wedge r = \text{sso}] \vee \exists S\, [\text{some\_none}_{(2,2)}(S, \text{add\_RO}, \text{rm\_RO}, 2, v, 3, r; 2, v, 3, r, T) \wedge \neg\text{in\_after}_1(S, \text{delete\_user}, 2, v, T)]]$$

$$\text{R}(v, T, r) \leftrightarrow [[v = \text{Root} \wedge r = \text{sso}] \vee \exists S\, [\text{some\_none}_{(2,2)}(S, \text{add\_R}, \text{rm\_R}, 2, v, 3, r; 2, v, 3, r, T) \wedge \neg\text{in\_after}_1(S, \text{delete\_user}, 2, v, T)]]$$

$$\text{CU}(v, T) = l \leftrightarrow [\exists S\, [\text{some\_none}_{(2,1)}(S, \text{create\_user}, \text{set\_cu}, 2, v, 3, l; 2, v, T) \wedge \neg\text{in\_after}_1(S, \text{delete\_user}, 2, v, T)] \vee [\exists S\, [\text{some\_none}(S, \text{set\_cu}, \text{set\_cu}, 2, v, 3, l; 2, v, T) \wedge \neg\text{in\_after}_1(S, \text{delete\_user}, 2, v, T)]]$$

$$\text{AS}(v, c, r, k, T) \leftrightarrow [\text{RO}(v, T, \text{sso}) \vee \exists S\, [\text{some\_none\_ref}_{(3,3)}(S, \text{add\_AS}, \text{rm\_AS}, (4, r), 2, v, 3, c, 5, k; (4, r), 2, v, 3, c, 5, k, T) \wedge \neg\text{in\_after}_1(S, \text{delete\_user}, 2, v, T)]]$$

$$\text{CE}(r, T) = l \leftrightarrow \exists S\, \exists C\, \exists s\, [[\text{callname}(C) = \text{set\_CE} \vee \text{callname}(C) = \text{cont\_create} \vee \text{callname}(C) = \text{obj\_create} \vee \text{callname}(C) = \text{downgrade}] \wedge \text{arg}(C, 2) = s \wedge \text{arg}(C, 3) = l \wedge \text{coref}(s, S, r, T) \wedge \neg\exists R\, \text{in\_after\_ref}_1(S.C.R, c, (2, r), 3, r, T) \wedge [c = \text{delete\_ref} \vee c = \text{downgrade} \vee c = \text{set\_CE}]$$

$$\text{CCR}(r, T) = \text{True} \leftrightarrow \exists C_0\, \exists R_0\, \forall C\, \forall R\, [[\text{callname}(C) = \text{set\_CCR} \wedge \text{prefix}(R.C, T) \wedge \text{coref}(\text{arg}(C, 2), R, r, T)] \leftrightarrow \text{prefix}(R.C, R_0.C_0)] \wedge \text{arg}(C, 3) = \text{True}]$$

$$\text{CCR}(r, T) = \text{False} \leftrightarrow \text{CCR}(r, T) \neq \text{True}$$

$$\text{T}(r, S) = x \leftrightarrow [x = \text{RM} \wedge \exists R\, \exists C\, [\forall T\, \forall D\, [[\text{prefix}(T, S) \wedge \text{callname}(D) = \text{delete\_ref} \wedge \text{coref}(\text{arg}(D, 2), T, r, S)] \leftrightarrow \text{prefix}(T.D, R.C)] \wedge \text{prefix}(R.C, S) \wedge \text{callname}(C) = \text{release} \wedge \text{coref}(\text{arg}(C, 2), R, r, S)]] \vee [x = \text{DM} \wedge \neg\exists R\, \exists C\, [\forall T\, \forall D\, [[\text{prefix}(T, S) \wedge \text{callname}(D) = \text{delete\_ref} \wedge \text{coref}(\text{arg}(D, 2), T, r, S)] \leftrightarrow \text{prefix}(T.D, R.C)] \wedge \text{prefix}(R.C, S) \wedge \text{callname}(C) = \text{release} \wedge \text{coref}(\text{arg}(C, 2), R, r, S)]]$$

$$\text{CD}(d, T) = x \leftrightarrow [\exists S\, \exists C\, [\text{callname}(C) = \text{set\_CD} \wedge x = \text{arg}(C, 3) \wedge \forall D\, \forall R\, [\text{prefix}(R.D, S.C) \leftrightarrow [\text{prefix}(R.D, T) \wedge \text{callname}(D) = \text{set\_CD} \wedge \text{coref}(\text{arg}(D, 2), R, d, T)]]] \vee [x = \text{unclassified} \wedge \neg\exists S\, \exists C\, [\text{callname}(C) = \text{set\_CD} \wedge \text{prefix}(S.C, T) \wedge \text{coref}(\text{arg}(C, 2), S, d, T)]]$$

$$\text{RE}(r, T, u) \leftrightarrow \exists S\, \exists C\, [\text{L}(T) \wedge \text{prefix}(S.C, T) \wedge \text{callname}(C) = \text{release} \wedge \text{arg}(C, 1) = u]$$

$$\text{H}(r, T, s, k) \leftrightarrow \text{coref}(r : k, T, s, T)$$

$$\text{D}(o, x, y, W) \leftrightarrow [\text{device}(o) \wedge [\exists u\, \exists r\, \exists f\, \exists T\, \exists U\, [y = f \wedge \text{nf}(U) \wedge W \equiv U \wedge \text{coref}(r, U, x, W) \wedge \text{prefix}(T.\text{display}(u, r, f, o), U) \wedge \text{some\_none\_ref}_{(1,1)}(T, \text{display}, \text{remove}, (2, r), 4, o; (2, r), 3, o, U)] \vee \exists u\, \exists N\, \exists f\, \exists T\, \exists U\, [x = N \wedge y = f \wedge \text{nf}(U) \wedge W \equiv U \wedge \text{prefix}(T.\text{identify}(u, N, f, o), U) \wedge \text{some\_none}_{(1,1)}(T, \text{identify}, \text{identify}, 4, o; 4, o, U)]]$$

Additional Axioms for Normal Form and Legality

$$\text{nf}(e)$$

$$\forall C\, (\text{L}(T.C) \rightarrow \text{L}(T))$$

$$\text{nf}(T) \rightarrow \text{L}(T)$$

$$\forall C\, (\text{nf}(T.C) \rightarrow \text{nf}(T))$$

END
DATE
10-88
DTIC